Luís Caires
Vasco T. Vasconcelos (Eds.)

LNCS 4703

CONCUR 2007 – Concurrency Theory

18th International Conference, CONCUR 2007
Lisbon, Portugal, September 2007
Proceedings

Springer

Luís Caires Vasco T. Vasconcelos (Eds.)

CONCUR 2007 – Concurrency Theory

18th International Conference, CONCUR 2007
Lisbon, Portugal, September 3-8, 2007
Proceedings

Springer

Volume Editors

Luís Caires
Universidade Nova de Lisboa
Portugal
E-mail: Luis.Caires@di.fct.unl.pt

Vasco T. Vasconcelos
Universidade de Lisboa
Portugal
E-mail: vv@di.fc.ul.pt

Library of Congress Control Number: 2007933184

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

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

ISSN 0302-9743
ISBN-10 3-540-74406-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-74406-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12112159 06/3180 5 4 3 2 1 0

Preface

This volume contains the proceedings of the 18th International Conference on Concurrency Theory, held at the Gulbenkian Foundation, Lisbon, Portugal, September 3–8, 2007. Since its first edition back in 1990, Concur has been a leading conference in concurrency theory, attracting researchers, graduate students, and practitioners, to exchange ideas, progresses, and challenges.

The Concur conference has traditionally covered a large spectrum of the field, including invited lecturers and tutorials by prominent researchers, contributed technical papers, and several associated workshops. Main topics include basic models and logics of concurrent and distributed computation (such as process algebras, Petri nets, domain theoretic or game theoretic models, modal and temporal logics), specialized models or classes of systems (such as synchronous systems, real time and hybrid systems, stochastic systems, databases, mobile and migrating systems, protocols, biologically inspired systems), related verification techniques and tools (including state-space exploration, model-checking, synthesis, abstraction, automated deduction, testing), and concurrent programming (such as distributed, constraint or object-oriented, graph rewriting, as well as associated type systems, static analyses, and abstract machines).

This year, the Program Committee, after a careful and thorough reviewing process, selected for inclusion in the programme 30 papers out of 112 submissions. Each submission was evaluated by at least three referees, and the accepted papers were selected during a one-and-a-half-week electronic discussion.

The volume opens with the invited contributions by Luca Aceto (Reykjavík), Vincent Danos (Paris 7), and Fred B. Schneider (Cornell). The program also included an addditional invited lecture by Peter O'Hearn (QMU London) and an additional tutorial by José Luiz Fiadeiro (Leicester), all of which, we believe, complimented the technical papers rather well.

Co-located with Concur 2007, eight workshops took place: Expressiveness in Concurrency (EXPRESS), Graph Transformation for Verification and Concurrency (GT-VC), Security Issues in Concurrency (SECCO), Verification and Analysis of Multi-threaded Java-like Programs (VAMP), Applying Concurrency Research in Industry (IFIP WG 1.8), Foundations of Coordination Languages and Software Architectures (FOCLASA), From Biology to Concurrency and Back (FBTC), and International Workshop on Verification of Infinite-State Systems (INFINITY).

Concur 2007 was made possible by the contribution and dedication of many people. First of all, we would like to thank all the authors who submitted papers for consideration. Secondly we would like to thank our invited and tutorial speakers. We would also like to thank the members of the Program Committee for their hard work, careful reviews, and the thorough and balanced discussions during the selection process. Finally, we acknowledge the Gulbenkian Foundation for hosting Concur 2007, and Easychair and Andrei Voronkov for providing software support for the Program Committee meeting.

June 2007

Luís Caires
Vasco T. Vasconcelos

Organization

Program Committee

Roberto Amadio, Université Paris 7, France
Jos Baeten, Eindhoven University of Technology, The Netherlands
Bruno Blanchet, Ecole Normale Supérieure de Paris, France
Franck van Breugel, York University, Canada
Luís Caires, Universidade Nova de Lisboa, Portugal (Co-chair)
Luca Cardelli, Microsoft Research Cambridge, UK
Luca de Alfaro, University of California at Santa Cruz, USA
Wan Fokkink, Free University of Amsterdam, The Netherlands
Daniel Hirschkoff, Ecole Normale Supérieure de Lyon, France
Radha Jagadeesan, DePaul University, USA
Alan Jeffrey, Bell Labs, Alcatel-Lucent, USA
Antonin Kucera, Masaryk University in Brno, Czech Republic
Faron Moller, University of Wales Swansea, UK
Ugo Montanari, University of Pisa, Italy
Uwe Nestmann, Technical University of Berlin, Germany
Mogens Nielsen, University of Aarhus, Denmark
Catuscia Palamidessi, INRIA Futurs Saclay and LIX, France
Davide Sangiorgi, Università di Bologna, Italy
Vladimiro Sassone, University of Southampton, UK
Peter Van Roy, Catholic University of Louvain, Belgium
Vasco T. Vasconcelos, Universidade de Lisboa, Portugal (Co-chair)
Hagen Völzer, IBM Research Zurich, Germany
Nobuko Yoshida, Imperial College London, UK

Additional Referees

Parosh Abdulla
Luca Aceto
Rajeev Alur
Roberto Amadio
Torben Amtoft
Henrik Reif Andersen
Jesus Aranda
Roland Axelsson
Eric Badouel
Martin Berger
Karthikeyan Bhargavan
Benedikt Bollig
Michele Boreale
Johannes Borgstrom
Tomáš Brázdil
Julian Bradfield
Sébastien Briais
Václav Brožek
Roberto Bruni
Marzia Buscemi
Nadia Busi
Marco Carbone
Antonio Cau
Pietro Cenciarelli
Bernadette Bost
Marsha Chechik
Taolue Chen
Tom Chothia
Raphael Collet
Ricardo Corin
Andrea Corradini
Flavio Corradini
Vincent Danos
Alexandre David
Pierpaolo Degano
Zoltan Esik

Marco Faella
Harald Fecher
Jérôme Feret
Rodrigo Ferreira
Wan Fokkink
Vojtěch Forejt
Cédric Fournet
Adrian Francalanza
Lars-Ake Fredlund
Xiang Fu
Rachele Fuzzati
Fabio Gadducci
Paul Gastin
Blaise Genest
Hugo Gimbert
Rob van Glabbeek
Ursula Goltz
Andy Gordon
Daniele Gorla
Stefan Haar
Peter Habermehl
Keijo Heljanko
Holger Hermanns
Claudio Hermida
André Hirschowitz
Yoram Hirshfeld
Jan Holeček
Kohei Honda
Hai Feng Huang
Alexei Iliasov
Petr Jančar
Yves Jaradin
Ranjit Jhala
Marcin Jurdzinski
Temesghen Kahsai
Ekkart Kindler
Bartek Klin
Alexander Knapp
Pavel Krcal
Jean Krivine
Jochen M. Kuester
Alexander Kurz
Dietrich Kuske
Celine Kuttler
Marta Kwiatkowska
Alberto Lluch Lafuente
Cosimo Laneve
Martin Lange
Ranko Lazic
Serguei Lenglet
Alexey Loginov
Markus Lohrey
Robert Lorenz
Gavin Lowe
Étienne Lozes
Gerald Luettgen
Yoad Lustig
Bas Luttik
Sergio Maffeis
Patrick Maier
Rupak Majumdar
Nicolas Markey
Jasen Markovski
Richard Mayr
Boris Mejias
Hernan Melgratti
Paul-André Mellies
Massimo Merro
Dale Miller
Robin Milner
Antoine Mine
Anca Muscholl
Francesco Zappa Nardelli
Dejan Nickovic
Rocco De Nicola
Carlos Olarte
James Ortiz
Simona Orzan
Prakash Panangaden
Radek Pelanek
Wojciech Penczek Anna
Philippou
Pierluigi San Pietro
Alberto Policriti
Roberto De Prisco
Rosario Pugliese
Frank Puhlmann
Luis Quesada
Julian Rathke
Arend Rensink
Willem-Paul de Roever
Brigitte Rozoy
Michal Rutkowski
Jan Rutten
Marko Samer
Alan Schmitt
Pierre-Yves Schobbens
Viktor Schuppan
Laura Semini
Natalia Sidorova
Pawel Sobocinski
Jiří Srba
Christian Stahl
Ian Stark
Martin Steffen
Marielle Stoelinga
Angelo Troina
Tachio Terauchi
Bent Thomsen
Alwen Tiu
Tayssir Touili
Nikola Trcka
Richard Trefler
Emilio Tuosto
Frank D. Valencia
Antti Valmari
Daniele Varacca
Björn Victor
Erik de Vink
Walter Vogler
Marc Voorhoeve
Jerome Vouillon
Peng Wu
Weirong Wang
Josef Widder
Karsten Wolf
Verena Wolf
James Worrell
Nobuko Yoshida
Gianluigi Zavattaro
Vojtěch Řehák
David Šafránek

Organization

Chairs

Luís Caires, Universidade Nova de Lisboa, Portugal
Vasco T. Vasconcelos, Universidade de Lisboa, Portugal

Workshops

Francisco Martins, Universidade de Lisboa, Portugal
António Ravara, Universidade Técnica de Lisboa, Portugal

Concur Steering Committee

Roberto Amadio, Université Paris 7, France
Jos Baeten, Eindhoven University of Technology, The Netherlands
Eike Best, University of Oldenburg, Germany
Kim Larsen, Aalborg University, Denmark
Ugo Montanari, University of Pisa, Italy
Scott Smolka, SUNY Stony Brook, USA

Sponsors

Fundação para a Ciência e Tecnologia, Ministério da Ciência e Ensino Superior
Centro de Informática e Tecnologias da Informação/FCT/UNL
Security and Quantum Information Group/Instituto de Telecomunicações
Câmara Municipal de Lisboa

Table of Contents

Mapping the Security Landscape: A Role for Language Techniques

Abstract of Invited Lecture

Fred B. Schneider[*]

Department of Computer Science
Cornell University
Ithaca, New York 14558
U.S.A
`fbs@cs.cornell.edu`

Over the last decade, programming language techniques have been applied in non-obvious ways to building secure systems. This talk will not only survey that work in *language based security* but show that the theoretical underpinnings of programming languages are a good place to start for developing a much needed foundation for software system security.

Research developments in language design, compilers, program analysis, type checking, and program rewriting have brought increased assurance in what a software system does and does not do. This was needed, welcome, but not surprising. What is surprising are the new approaches to engineering secure systems that have emerged from language-based security research. Inline reference monitors, for example, enable enforcement of a broad class of authorization policies specified separately from the components they concern; and proof carrying code provides a way to relocate trust from one system component to another, providing designers with flexibility in setting the trusted computing base.

Our sophistication in the engineering of secure systems has improved without the benefit of a rigorous mathematical foundation. This is probably a matter of luck and certainly a matter for concern. Too much of security engineering is based on anecdotes, driven by past attacks, and directed by a few familiar kinds of security policies. We have still to identify abstract classes of enforcement mechanisms, attacks, or policies, much less understand connections between them. In short, we have made little progress in mapping the security landscape.

Yet some features of the security landscape do start to become apparent if the abstractions and metaphors of programming language semantics are applied to software system security. And there is reason to believe the approach will yield more fruit, too. We will discuss what is known, what we might strive to know, and the role that programming language semantics can play.

Acknowledgment. I am grateful to my collaborators Michael Clarkson, Kevin Hamlen, Andrew Myers, Greg Morrisett, and Riccardo Pucella.

[*] Supported in part by AFOSR grant F9550-06-0019, National Science Foundation Grants 0430161 and CCF-0424422 (TRUST), and a gift from Microsoft Corporation.

The Saga of the Axiomatization of Parallel Composition[*]

Luca Aceto and Anna Ingolfsdottir

Department of Computer Science, Reykjavík University
Kringlan 1, 103 Reykjavík, Iceland
luca@ru.is, annai@ru.is

Abstract. This paper surveys some classic and recent results on the finite axiomatizability of bisimilarity over CCS-like languages. It focuses, in particular, on non-finite axiomatizability results stemming from the semantic interplay between parallel composition and nondeterministic choice. The paper also highlights the role that auxiliary operators, such as Bergstra and Klop's left and communication merge and Hennessy's merge operator, play in the search for a finite, equational axiomatization of parallel composition both for classic process algebras and for their real-time extensions.

1 The Problem and Its History

Process algebras are prototype description languages for reactive systems that arose from the pioneering work of figures like Bergstra, Hoare, Klop and Milner. Well-known examples of such languages are ACP [18], CCS [44], CSP [40] and Meije [13]. These algebraic description languages for processes differ in the basic collection of operators that they offer for building new process descriptions from existing ones. However, since they are designed to allow for the description and analysis of systems of interacting processes, all these languages contain some form of *parallel composition* (also known as *merge*) operator allowing one to put two process terms in parallel with one another. These operators usually interleave the behaviours of their arguments, and support some form of synchronization between them.

For example, Milner's CCS offers the binary operator $||$, whose intended semantics is described by the following classic rules in the style of Plotkin [49].

$$\frac{x \xrightarrow{\mu} x'}{x \,||\, y \xrightarrow{\mu} x' \,||\, y} \qquad \frac{y \xrightarrow{\mu} y'}{x \,||\, y \xrightarrow{\mu} x \,||\, y'} \qquad \frac{x \xrightarrow{\alpha} x',\ y \xrightarrow{\bar{\alpha}} y'}{x \,||\, y \xrightarrow{\tau} x' \,||\, y'} \tag{1}$$

(In the above rules, the symbol μ stands for an action that a process may perform, α and $\bar{\alpha}$ are two observable actions that may synchronize, and τ is a symbol denoting the result of their synchronization.)

[*] This paper is based on joint work with Wan Fokkink, Bas Luttik and MohammadReza Mousavi. The authors were partly supported by the project "The Equational Logic of Parallel Processes" (nr. 060013021) of The Icelandic Research Fund.

Although the above rules describe the behaviour of the parallel composition operator in very intuitive fashion, the equational characterization of this operator is not straightforward. In their seminal paper [39], Hennessy and Milner offered, amongst a wealth of other classic results, a complete equational axiomatization of bisimulation equivalence [48] over the recursion-free fragment of CCS. (See the paper [14] for a more detailed historical account highlighting, e.g., Hans Bekić's early contributions to this field of research.) The axiomatization proposed by Hennessy and Milner in [39] dealt with parallel composition using the so-called *expansion law*—a law that, intuitively, allows one to obtain a term describing explicitly the initial transitions of the parallel composition of two terms whose initial transitions are known. This law can be expressed as the following equation schema

$$\Big(\sum_{i\in I}\mu_i x_i\Big) \,||\, \Big(\sum_{j\in J}\gamma_j y_j\Big) = \sum_{i\in I}\mu_i(x_i \,||\, y) + \sum_{j\in J}\gamma_j(x \,||\, y_j) + \sum_{\substack{i\in I, j\in J\\ \mu_i=\overline{\gamma_j}}}\tau(x_i \,||\, y_j) \quad (2)$$

(where I and J are two finite index sets, and the μ_i and γ_j are actions), and is nothing but an equational formulation of the aforementioned rules describing the operational semantics of parallel composition.

Despite its natural and simple formulation, the expansion law, however, is an equation schema with a countably infinite number of instances. This raised the question of whether the parallel composition operator could be axiomatized in bisimulation semantics by means of a finite collection of equations. This question was answered positively by Bergstra and Klop, who gave in [20] a finite equational axiomatization of the merge operator in terms of the auxiliary left merge and communication merge operators. Moller showed in [46,47] that bisimulation equivalence is not finitely based over CCS and PA without the left merge operator. (The process algebra PA [20] contains a parallel composition operator based on pure interleaving without communication—viz. an operator described by the first two rules in (1)—and the left merge operator.) These results, which we survey in Section 2, indicate that auxiliary operators are necessary to obtain a finite axiomatization of parallel composition.

Moller's results clarified the role played by the expansion law in the equational axiomatization of parallel composition over CCS and, to the best of our knowledge, were the first negative results on the existence of finite equational axiomatizations for algebras of processes that were presented in the literature. To our mind, the negative results achieved by Moller in his doctoral dissertation removed a psychological barrier by showing that non-finite axiomatizability results could indeed be achieved also in the study of process algebras, and paved the way to the further developments we describe henceforth in this paper.

The contributions our collaborators and we have offered so far to the saga of the axiomatization of parallel composition have been mostly motivated by an attempt to answer the following questions.

1. Are there other "natural" auxiliary operators that can be used, in lieu of Bergstra and Klop's left and communication merge, to achieve a finite equational axiomatization of parallel composition?

2. Do the aforementioned results hold true also for extensions of classic process algebras like CCS with features such as real-time?

As far as the former motivating question is concerned, the literature on process algebra offers at least one alternative proposal to the use of the left and communication merge operators. In the paper [38], which we believe is not so well known as it deserves to be, Hennessy proposed an axiomatization of observation congruence [39] and split-2 congruence over a CCS-like recursion-free process language. (It is worth noting for the sake of historical accuracy that the results reported in [38] were actually obtained in 1981; see the preprint [36].) Those axiomatizations used an auxiliary operator, denoted $|\!/$ by Hennessy, that is essentially a combination of the left and communication merge operators as its behaviour is described by the first and the last rule in (1). Apart from having soundness problems (see the reference [2] for a general discussion of this problem, and corrected proofs of Hennessy's results), the proposed axiomatization of observation congruence offered in [38] is *infinite*, as it uses a variant of the expansion law from [39]. This led Bergstra and Klop to write in [20, page 118] that:

"It seems that γ does not have a finite equational axiomatization."

(In [20] Bergstra and Klop used γ to denote Hennessy's merge.) In Section 3, we will present an answer to this conjecture of Bergstra and Klop's by showing that, in the presence of two distinct complementary actions, it is impossible to provide a finite axiomatization of the recursion-free fragment of CCS modulo bisimulation equivalence using Hennessy's merge operator $|\!/$. We believe that this result, which was originally proved in [6], further reinforces the status of the left merge and the communication merge operators as auxiliary operators in the finite equational characterization of parallel composition in bisimulation semantics. Interestingly, as shown in [8], in sharp contrast to the situation in standard bisimulation semantics, CCS with Hennessy's merge *can be finitely axiomatized* modulo split-2 bisimulation equivalence [33,38]. (Split-2 bisimilarity is defined like standard bisimilarity, but is based on the assumption that action occurrences have a beginning and an ending, and that these events may be observed.) This shows that, in sharp contrast to the results offered in [45,46], "reasonable congruences" finer than standard bisimulation equivalence can be finitely axiomatized over CCS using Hennessy's merge as the single auxiliary operation—compare with the non-finite axiomatizability results for these congruences offered in [45,46].

It is also natural to ask oneself whether the aforementioned non-finite axiomatizability results hold true also for extensions of the basic CCS calculus with features such as real-time. In Section 4, we review some negative results, originally shown in [12], on the finite axiomatizability of timed bisimilarity over Yi's timed CCS [52,53]. In particular, we prove that timed bisimilarity is not finitely based both for single-sorted and two-sorted presentations of timed CCS. We further strengthen this result by showing that, unlike in the setting of CCS, adding the untimed or the timed left merge operator to the syntax and semantics of timed CCS does not solve the axiomatizability problem. To our mind, these results indicate that the expressive power that is gained by adding to CCS linguistic features suitable for the description of timing-based behaviours substantially complicates the equational theory of the resulting algebras of processes.

We feel that there are still many chapters to be written in the saga of the study of the equational logic of parallel composition, and we list a few open problems and directions of ongoing research throughout this paper.

Related Work in Concurrency and Formal Language Theory. The equational characterization of different versions of the parallel composition operator is a classic topic in the theory of computation. In particular, the process algebraic literature abounds with results on equational axiomatizations of various notions of behavioural equivalence or preorder over languages incorporating some notion of parallel composition—see, e.g., the textbooks [18,30,37,44] and the classic papers [20,39,43] for general references. Early ω-complete axiomatizations are offered in [35,45]. More recently, Fokkink and Luttik have shown in [31] that the process algebra PA [20] affords an ω-complete axiomatization that is finite if so is the underlying set of actions. As shown in [9], the same holds true for the fragment of CCS without recursion, relabelling and restriction extended with the left and communication merge operators. The readers will find a survey of recent results on the equational logic of processes in [7], and further non-finite axiomatizability results for rather basic process algebras in, e.g., [4,10].

An analysis of the reasons why operators like the left merge and the communication merge are equationally well behaved in bisimulation semantics has led to general algorithms for the generation of (finite) equational axiomatizations for behavioural equivalences from various types of transition system specifications—see, e.g., [1,3,15] and the references in [11] for further details.

Parallel composition appears as the shuffle operator in the time-honoured theory of formal languages. Not surprisingly, the equational theory of shuffle has received considerable attention in the literature. Here we limit ourselves to mentioning some results that have a close relationship with process theory.

In [51], Tschantz offered a finite equational axiomatization of the theory of languages over concatenation and shuffle, solving an open problem raised by Pratt. In proving this result he essentially rediscovered the concept of pomset [34,50]—a model of concurrency based on partial orders whose algebraic aspects have been investigated by Gischer in [32]—, and proved that the equational theory of series-parallel pomsets coincides with that of languages over concatenation and shuffle. The argument adopted by Tschantz in his proof was based on the observation that series-parallel pomsets may be coded by a suitable homomorphism into languages, where the series and parallel composition operators on pomsets are modelled by the concatenation and shuffle operators on languages, respectively. Tschantz's technique of coding pomsets with languages homomorphically was further extended in the papers [22,24,25] to deal with several other operators, infinite pomsets and infinitary languages, as well as sets of pomsets. The axiomatizations by Gischer and Tschantz have later been extended in [25,29] to a two-sorted language with ω-powers of the concatenation and parallel composition operators. The axiomatization of the algebra of pomsets resulting from the addition of these iteration operators is, however, necessarily infinite because, as shown in [29], no finite collection of equations can capture all the sound equalities involving them.

The results of Moller's on the non-finite axiomatizability of bisimulation equivalence over the recursion-free fragment of CCS and PA without the left merge operator given in [46,47] are paralleled in the world of formal language theory by those offered in [21,23,28]. In the first of those references, Bloom and Ésik proved that the valid inequations in the algebra of languages equipped with concatenation and shuffle have no finite basis. Ésik and Bertol showed in [28] that the equational theory of union, concatenation and shuffle over languages has no finite first-order axiomatization relative to the collection of all valid inequations that hold for concatenation and shuffle. Hence the combination of some form of parallel composition, sequencing and choice is hard to characterize equationally both in the theory of languages and in that of processes. Moreover, Bloom and Ésik have shown in [23] that the variety of all languages over a finite alphabet ordered by inclusion with the operators of concatenation and shuffle, and a constant denoting the singleton language containing only the empty word, is not finitely axiomatizable by first-order sentences that are valid in the equational theory of languages over concatenation, union and shuffle.

2 Background

The core process algebra that we shall consider henceforth in this paper is a fragment of Milner's CCS. This language, which will be referred to simply as CCS, is given by the following grammar:

$$t ::= x \quad | \quad \mathbf{0} \quad | \quad at \quad | \quad \bar{a}t \quad | \quad \tau t \quad | \quad t+t \quad | \quad t \,||\, t \ ,$$

where x is a variable drawn from a countably infinite set V, a is an action, and $\bar{a}$ is its complement. We assume that the actions a and $\bar{a}$ are distinct. Following Milner [44], the action symbol τ will result from the synchronized occurrence of the complementary actions a and $\bar{a}$. We let $\mu \in \{a, \bar{a}, \tau\}$ and $\alpha \in \{a, \bar{a}\}$. (We remark, in passing, that this small collection of actions suffices to prove all the negative results we survey in this study. All the positive results we shall present in what follows hold for arbitrary finite sets of actions.) As usual, we postulate that $\bar{\bar{a}} = a$. We shall use the meta-variables t, u to range over process terms. The *size* of a term is the number of operator symbols in it. A process term is *closed* if it does not contain any variables. Closed terms will be typically denoted by p, q.

In the remainder of this paper, we let a^0 denote $\mathbf{0}$, and a^{m+1} denote $a(a^m)$. We sometimes simply write a in lieu of a^1.

The SOS rules for the above language are standard, and may be found in Table 1. These transition rules give rise to transitions between closed terms. The operational semantics for our language, and for all its extensions that we shall introduce in the remainder of this paper, is thus given by a labelled transition system [42] whose states are closed terms, and whose labelled transitions are

Table 1. SOS Rules for the CCS Operators ($\mu \in \{a, \bar{a}, \tau\}$ and $\alpha \in \{a, \bar{a}\}$)

$$\frac{}{\mu x \xrightarrow{\mu} x} \qquad \frac{x \xrightarrow{\mu} x'}{x + y \xrightarrow{\mu} x'} \qquad \frac{y \xrightarrow{\mu} y'}{x + y \xrightarrow{\mu} y'}$$

$$\frac{x \xrightarrow{\mu} x'}{x \,||\, y \xrightarrow{\mu} x' \,||\, y} \qquad \frac{y \xrightarrow{\mu} y'}{x \,||\, y \xrightarrow{\mu} x \,||\, y'} \qquad \frac{x \xrightarrow{\alpha} x',\ y \xrightarrow{\bar{\alpha}} y'}{x \,||\, y \xrightarrow{\tau} x' \,||\, y'}$$

those that are provable using the rules that are relevant for the language under consideration.

In this paper, we shall consider our core language and all its extensions modulo bisimulation equivalence [44,48].

Definition 1. *Bisimulation equivalence (also sometimes referred to as bisimilarity), denoted by $\underline{\leftrightarrow}$, is the largest symmetric relation over closed terms such that whenever $p \,\underline{\leftrightarrow}\, q$ and $p \xrightarrow{\mu} p'$, then there is a transition $q \xrightarrow{\mu} q'$ with $p' \,\underline{\leftrightarrow}\, q'$.*
If $p \,\underline{\leftrightarrow}\, q$, then we say that p and q are bisimilar.

It is well known that, as its name suggests, bisimulation equivalence is indeed an equivalence relation (see, e.g., the references [44,48]). Since the SOS rules in Table 1 (and all of the other rules we shall introduce in the remainder of this paper) are in de Simone's format [27], bisimulation equivalence is a congruence.

Bisimulation equivalence is extended to arbitrary terms in the standard way.

2.1 Classic Results on Equational Axiomatizations

An *axiom system* is a collection of equations $t \approx u$, where t and u are terms. An equation $t \approx u$ is derivable from an axiom system E if it can be proved from the axioms in E using the rules of equational logic (viz. reflexivity, symmetry, transitivity, substitution and closure under contexts). An equation $t \approx u$ is *sound* with respect to $\underline{\leftrightarrow}$ iff $t \,\underline{\leftrightarrow}\, u$. An axiom system is sound with respect to $\underline{\leftrightarrow}$ iff so is each of its equations. For example, the axiom system in Table 2 is sound. In what follows, we use a *summation* $\sum_{i \in \{1,\ldots,k\}} t_i$ to denote $t_1 + \cdots + t_k$, where the empty sum represents $\mathbf{0}$.

An axiom system E is a *complete axiomatization* of $\underline{\leftrightarrow}$ over (some extension of) CCS if E is sound with respect to $\underline{\leftrightarrow}$, and proves all of the equations over the language that are sound with respect to $\underline{\leftrightarrow}$. If E is sound with respect to $\underline{\leftrightarrow}$, and proves all of the *closed* equations over the language that are sound with respect to $\underline{\leftrightarrow}$, then we say that E is *ground complete*.

The study of equational axiomatizations of bisimilarity over process algebras was initiated by Hennessy and Milner, who proved the following classic result.

Theorem 1 (Hennessy and Milner [39]). *The axiom system consisting of equations* A1–A4 *and all of the instances of (2) is ground complete for bisimilarity over* CCS.

Since the equation schema (2) has infinitely many instances, the above theorem raised the question of whether the parallel composition operator could be axiomatized in bisimulation semantics by means of a finite collection of equations.

Table 2. Some Axioms for Bisimilarity

A1	$x + y \approx y + x$
A2	$(x + y) + z \approx x + (y + z)$
A3	$x + x \approx x$
A4	$x + \mathbf{0} \approx x$

This question was answered positively by Bergstra and Klop, who gave in [20] a finite ground-complete axiomatization of the merge operator in terms of the auxiliary left merge and communication merge operators. The operational rules for these operators are

$$\frac{x \xrightarrow{\mu} x'}{x \mathbin{\lfloor\!\lfloor} y \xrightarrow{\mu} x' \,||\, y} \qquad \frac{x \xrightarrow{\alpha} x',\; y \xrightarrow{\bar{\alpha}} y'}{x \,|\, y \xrightarrow{\tau} x' \,||\, y'}$$

where $\lfloor\!\lfloor$ and $|$ stand for the left and communication merge operators, respectively.

But, are auxiliary operators necessary to obtain a finite equational axiomatization of bisimilarity over the language CCS? This question remained unanswered for about a decade until Moller proved the following seminal result in his doctoral dissertation.

Theorem 2 (Moller [45,47]). *Bisimilarity has no finite, (ground-)complete equational axiomatization over* CCS*.*

Thus auxiliary operators are indeed necessary to obtain a finite axiomatization of parallel composition, and the expansion law cannot be replaced by a finite collection of sound equations.

Moller's proof of the theorem above is based on the observation that, since $||$ does not distribute over $+$, no finite, sound axiom system over CCS can be powerful enough to "expand" the initial behaviour of a term of the form $a \,||\, p$ when p has a sufficiently large number of initial transitions leading to non-bisimilar terms. It follows that no finite collection of sound axioms is as powerful as the expansion law (2). Technically, Moller showed that, when n is greater than the size of the largest term in a finite, sound axiom system E over the language CCS, E cannot prove the sound equation

$$a \,||\, \sum_{i=1}^{n} a^i \approx a(\sum_{i=1}^{n} a^i) + \sum_{i=2}^{n+1} a^i \ .$$

Note that, up to bisimilarity, the right-hand side of the above equation expresses "syntactically" the collection of initial transitions of the term on the left-hand side.

Remark 1. Theorem 2 holds true for each "reasonable congruence" over CCS. A congruence is "reasonable" in the sense of Moller if it is included in bisimilarity and satisfies the family of equations Red_n presented in [45, page 111].

3 The Role of Hennessy's Merge

Theorem 2 shows that one cannot hope to achieve a finite (ground-)complete axiomatization for bisimilarity over CCS without recourse to auxiliary operators. Moreover, the work by Bergstra and Klop presented in [20] tells us that a finite ground-complete axiomatization can be obtained at the price of adding the left and communication merge operators to the language. (In fact, as shown in [9], for any finite set of actions the resulting language also affords a finite *complete* axiomatization modulo bisimilarity.) A natural question to ask at this point is whether one can obtain a finite equational axiomatization of bisimilarity over CCS extended with some auxiliary binary operator other than those proposed by Bergstra and Klop. An independent proposal, put forward by Hennessy in [36,38], is to add the auxiliary operator $|\!/$ with the following SOS rules to the signature for CCS.

$$\frac{x \xrightarrow{\mu} x'}{x \,|\!/\, y \xrightarrow{\mu} x' \,||\, y} \qquad \frac{x \xrightarrow{\alpha} x',\ y \xrightarrow{\bar{\alpha}} y'}{x \,|\!/\, y \xrightarrow{\tau} x' \,||\, y'}$$

Note that the above operator is essentially a combination of the left and communication merge operators. We denote the resulting language by CCS_H.

Does bisimilarity afford a finite equational axiomatization over CCS_H? In [20, page 118], Bergstra and Klop conjectured a negative answer to the above question. Their conjecture was finally confirmed about twenty years later by the following theorem.

Theorem 3 (Aceto, Fokkink, Ingolfsdottir and Luttik [6]). *Bisimulation equivalence admits no finite (ground-)complete equational axiomatization over the language* CCS_H*.*

The aforementioned negative result holds in a very strong form. Indeed, we prove that no finite collection of equations over CCS_H that are sound with respect to bisimulation equivalence can prove all of the sound closed equalities of the form

$$e_n: \quad a \,|\!/\, p_n \approx a p_n + \sum_{i=0}^{n} \tau a^i \quad (n \geq 0) \ ,$$

where the terms p_n are defined thus:

$$p_n = \sum_{i=0}^{n} \bar{a} a^i \quad (n \geq 0) \ .$$

The proof of Theorem 3 is given along proof-theoretic lines that have their roots in Moller's proof of Theorem 2. However, the presence of possible synchronizations in the terms used in the family of equations e_n is necessary for our result, and requires careful attention in our proof. (Indeed, in the absence of synchronization, Hennessy's merge reduces to Bergstra and Klop's left merge operator, and thus affords a finite equational axiomatization.) In particular, the infinite family of equations e_n and our arguments based upon it exploit the inability of any finite axiom system E that is sound with respect to bisimulation equivalence to "expand" the synchronization behaviour of terms of the form $p \mathbin{|\!/} q$, for terms q that, like the terms p_n above eventually do, have a number of inequivalent "summands" that is larger than the maximum size of the terms mentioned in equations in E. As in the original arguments of Moller's, the root of this problem can be traced back to the fact that, since $\mathbin{|\!/}$ distributes with respect to the choice operator $+$ in the first argument but *not* in the second, no finite collection of equations can express the interplay between interleaving and communication that underlies the semantics of Hennessy's merge.

Our Theorem 3 is the counterpart of Moller's Theorem 2 over the language CCS_H. As we recalled in Remark 1, Moller's non-finite axiomatizability result for CCS holds for each "reasonable" congruence. It is therefore natural to ask ourselves whether each "reasonable" congruence is not finitely based over CCS_H too. The following result shows that, in sharp contrast to the situation in standard bisimulation semantics, the language CCS_H *can be finitely axiomatized* modulo split-2 bisimulation equivalence [36,38], and therefore that, modulo this non-interleaving equivalence, the use of Hennessy's merge suffices to yield a finite axiomatization of the parallel composition operation.

Theorem 4 (Aceto, Fokkink, Ingolfsdottir and Luttik [8]). *Split-2 bisimilarity affords a finite ground-complete equational axiomatization over the language* CCS_H*.*

The above result hints at the possibility that non-interleaving equivalences like split-2 bisimilarity may be finitely axiomatizable using a single binary auxiliary operator. Whether a similar result holds true for standard bisimilarity remains open. We conjecture that the use of *two* binary auxiliary operators is necessary to achieve a finite axiomatization of parallel composition in bisimulation semantics. This result would offer the definitive justification we seek for the canonical standing of the auxiliary operators proposed by Bergstra and Klop. Preliminary work on the confirmation of some form of this conjecture is under way [5].

4 The Influence of Time

So far in this paper we have presented, mostly negative, results on the finite axiomatizability of notions of bisimilarity over variations on Milner's CCS. Over the years, several extensions of CCS with, e.g., time, probabilities and priority have been presented in the literature. However, to the best of our knowledge, the question whether the aforementioned negative results hold true also for these extensions of classic process algebras like CCS has not received much attention in the research literature. In what follows, we discuss some impossibility results in the equational logic of timed bisimilarity over a fragment of Yi's timed CCS (TCCS) [52,53], which is one of the best-known timed extension of Milner's CCS.

One of the first design decisions to be taken when developing a language for the description of timing-based behaviours is what structure to use to model time. Since we are interested in studying the equational theory of TCCS modulo bisimilarity, rather than selecting a single mathematical structure, such as the natural numbers or the non-negative rationals or reals, to represent time, we feel that it is more satisfying to adopt an axiomatic approach. We will therefore axiomatize a class of mathematical models of time for which our negative results hold. The non-negative rationals and the non-negative reals will be specific instances of our axiomatic framework, amongst others.

Following [41], we define a monoid $(X, +, 0)$ to be:

– *left-cancellative* iff $(x + y = x + z) \Rightarrow (y = z)$, and
– *anti-symmetric* iff $(x + y = 0) \Rightarrow (x = y = 0)$.

We define a partial order on X as $x \leq y$ iff $x + z = y$ for some $z \in X$. A *time domain* is a left-cancellative anti-symmetric monoid $(D, +, 0)$ such that $\leq$ is a total order. A time domain is *non-trivial* if D contains at least two elements. Note that every non-trivial time domain does not have a largest element, and is therefore infinite. A time domain has 0 as *cluster point* iff for each $d \in D$ such that $d \neq 0$ there is a $d' \in D$ such that $0 < d' < d$. In what follows, we assume that our time domain, denoted henceforth by D, is non-trivial and has 0 as cluster point.

Syntactically, we consider the language TCCS that is obtained by adding to the signature of CCS delay prefixing operators of the form $\epsilon(d).\_$, where d is a non-zero element of a time domain D. *In what follows, we only consider action prefixing operators of the form at and parallel composition without synchronization.*

The operational semantics for closed TCCS terms is based on two types of transition relations: $\xrightarrow{a}$ for action transitions and $\xrightarrow{d}$, where $d \in D$, for time-delay transitions. Action transitions are defined by the rules in Table 1, whereas the Plotkin-style rules defining delay transitions are given below.

$$\frac{}{\mathbf{0} \xrightarrow{d} \mathbf{0}} \qquad \frac{}{ax \xrightarrow{d} ax}$$

$$\frac{}{\epsilon(d).x \xrightarrow{d} x} \qquad \frac{}{\epsilon(d+e).x \xrightarrow{d} \epsilon(e).x} \qquad \frac{x \xrightarrow{e} y}{\epsilon(d).x \xrightarrow{d+e} y}$$

$$\frac{x_0 \xrightarrow{d} y_0 \quad x_1 \xrightarrow{d} y_1}{x_0 + x_1 \xrightarrow{\epsilon(d)} y_0 + y_1} \qquad \frac{x_0 \xrightarrow{d} y_0 \quad x_1 \xrightarrow{d} y_1}{x_0 \,||\, x_1 \xrightarrow{d} y_0 \,||\, y_1}$$

The notion of equivalence over TCCS that we are interested in is *timed bisimilarity*. This is defined exactly as in Definition 1, with the proviso that the meta-variable μ now ranges over time delays as well as actions. For example, $\mathbf{0}$ and $\epsilon(d).\mathbf{0}$ are timed bisimilar for each d, and so are a and $a + \epsilon(d).a$. On the other hand, a and $\epsilon(d).a$ are not timed bisimilar because the former term affords an a-labelled transition whereas the latter does not. (Intuitively, the latter term has to wait for d units of time before being able to perform the action a.)

It is natural to wonder whether TCCS affords a finite (ground-)complete axiomatization modulo timed bisimilarity. Before addressing this question, let us remark that one can take two different approaches to formalizing the syntax of TCCS in a term algebra.

1. The first approach is to use a single-sorted algebra with the only available sort representing processes. Then $\epsilon(d).\_$ is a set of unary operators, one for each $d \in D$.
2. The other approach is to take two different sorts, one for time and one for processes, denoted by $\mathbb{T}$ and $\mathbb{P}$, respectively. Then, $\epsilon(\_)$ is a single function symbol with arity $\mathbb{T} \times \mathbb{P} \to \mathbb{P}$.

If we decide to follow the first approach then, since our time domain is infinite, we are immediately led to observe that no finite collection of sound equations can prove all of the valid equalities of the form $\mathbf{0} \approx \epsilon(d).\mathbf{0}$. As a corollary of this observation, we obtain the following result.

Theorem 5 (Aceto, Ingolfsdottir and Mousavi [12]). *Timed bisimilarity over single-sorted* TCCS *has no finite (ground-)complete axiomatization.*

The lesson to be drawn from the above result is that, in the presence of an infinite time domain, when studying the equational theory of TCCS, it is much more natural to consider a two-sorted presentation of the calculus. However, even in a two-sorted setting, we are still faced with the inability of any finite sound axiom system to capture the interplay between interleaving and non-determinism, which underlies Theorem 2. Therefore, by carefully adapting the proof of Moller's result, we obtain the following theorem.

Theorem 6 (Aceto, Ingolfsdottir and Mousavi [12]). *Timed bisimilarity over two-sorted* TCCS *has no finite (ground-)complete axiomatization.*

As shown by Bergstra and Klop in [19], in the setting of classic CCS and in the absence of synchronization one can finitely axiomatize bisimilarity by adding the left merge operator to the syntax for CCS. It is therefore natural to ask ourselves whether a finite axiomatization of timed bisimilarity over the fragment of TCCS we consider in this study can be obtained by adding some version of the left merge operator to the syntax for TCCS. Our order of business will now be to show that, unlike in the setting of Milner's CCS, even adding two variations on the left merge operator does not improve the situation with respect to axiomatizability.

We begin by noting that adding the classic left merge operator proposed by Bergstra and Klop to the syntax of TCCS does not lead to a axiomatizable theory.

Theorem 7 (Aceto, Ingolfsdottir and Mousavi [12]). *Timed bisimilarity over two-sorted* TCCS *extended with Bergstra and Klop's left merge operator has no finite (ground-)complete axiomatization.*

Following the tradition of Bergstra and Klop, the left merge operator was given a timed semantics in [17] as follows.

$$\frac{x_0 \xrightarrow{a} y_0}{x_0 \mathbin{|\!\!\lfloor} x_1 \xrightarrow{a} y_0 \mid\mid x_1} \qquad \frac{x_0 \xrightarrow{d} y_0 \quad x_1 \xrightarrow{d} y_1}{x_0 \mathbin{|\!\!\lfloor} x_1 \xrightarrow{d} y_0 \mathbin{|\!\!\lfloor} y_1}$$

This timed left merge operator enjoys most of the axioms for the classic left merge operator. However, this operator does not help in obtaining a finite ground-complete axiomatization for TCCS modulo bisimilarity either.

Theorem 8 (Aceto, Ingolfsdottir and Mousavi [12]). *Two-sorted* TCCS *extended with the timed left merge operator affords no finite (ground-)complete axiomatization modulo timed bisimilarity.*

Intuitively, the reason for the above-mentioned negative result is that the axiom

$$(ax) \mathbin{|\!\lfloor} y \approx a(x \,||\, y) \enspace ,$$

which is sound in the untimed setting, is in general unsound over TCCS. For example, consider the process $a \mathbin{|\!\lfloor} \epsilon(d).a$; after making a time delay of length d, it results in $a \mathbin{|\!\lfloor} a$, which is capable of performing two consecutive a-transitions. On the other hand, $a(\mathbf{0} \,||\, \epsilon(d).a)$ after a time delay of length d remains the same process and can only perform one a-transition, since the second a-transition still has to wait for d time units before becoming enabled.

However, the above axiom *is sound* for a class of TCCS processes whose behaviour, modulo timed bisimilarity, does not change by delaying. For instance, we have that

$$a \mathbin{|\!\lfloor} \left(\sum_{i=1}^{n} a \left(\sum_{j=1}^{i} a^j \right) \right) \leftrightarroweq a \left(\sum_{i=1}^{n} a \left(\sum_{j=1}^{i} a^j \right) \right)$$

for each $n \geq 0$. However, no finite sound axiom system can prove all of the above equalities, and therefore cannot be complete.

Remark 2. All of the impossibility results presented in this section also hold for conditional equations of the form $P \Rightarrow t \approx u$, where P is an arbitrary predicate over the time domain, and t, u are TCCS terms.

In the case of two-sorted TCCS, our proofs make use of the fact that the time domain has 0 as a cluster point. However, we conjecture that discrete-time TCCS, or its extension with (timed) left merge, is not finitely axiomatizable modulo timed bisimilarity either. Work on a proof of this conjecture is ongoing.

References

1. Aceto, L.: Deriving complete inference systems for a class of GSOS languages generating regular behaviours. In: Jonsson, B., Parrow, J. (eds.) CONCUR 1994. LNCS, vol. 836, pp. 449–464. Springer, Heidelberg (1994)
2. Aceto, L.: On Axiomatising finite concurrent processes. SIAM J. Comput. 23(4), 852–863 (1994)
3. Aceto, L., Bloom, B., Vaandrager, F.: Turning SOS rules into equations. Information and Computation 111(1), 1–52 (1994)
4. Aceto, L., Chen, T., Fokkink, W., Ingolfsdottir, A.: On the axiomatizability of priority. In: Bugliesi et al. [26], pp. 480–491

5. Aceto, L., Fokkink, W., Ingolfsdottir, A., Luttik, B.: Are two binary operators necessary to finitely axiomatize parallel composition? (in preparation)
6. Aceto, L., Fokkink, W., Ingolfsdottir, A., Luttik, B.: CCS with Hennessy's merge has no finite equational axiomatization. Theoretical Comput. Sci. 330(3), 377–405 (2005)
7. Aceto, L., Fokkink, W., Ingolfsdottir, A., Luttik, B.: Finite equational bases in process algebra: Results and open questions. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R.C. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity. LNCS, vol. 3838, pp. 338–367. Springer, Heidelberg (2005)
8. Aceto, L., Fokkink, W., Ingolfsdottir, A., Luttik, B.: Split-2 bisimilarity has a finite axiomatization over CCS with Hennessy's merge. Logical Methods in Computer Science 1(1), 1–12 (2005)
9. Aceto, L., Fokkink, W., Ingolfsdottir, A., Luttik, B.: A finite equational base for CCS with left merge and communication merge. In: Bugliesi et al. [26], pp. 492–503
10. Aceto, L., Fokkink, W., Ingolfsdottir, A., Nain, S.: Bisimilarity is not finitely based over BPA with interrupt. Theoretical Comput. Sci. 366(1–2), 60–81 (2006)
11. Aceto, L., Fokkink, W., Verhoef, C.: Structural operational semantics. In: Handbook of Process Algebra, pp. 197–292. North-Holland, Amsterdam (2001)
12. Aceto, L., Ingolfsdottir, A., Mousavi, M.: Impossibility results for the equational theory of timed CCS. In: Proceedings of the 2nd Conference on Algebra and Coalgebra in Computer Science. LNCS, Springer, Heidelberg (2007)
13. Austry, D., Boudol, G.: Algèbre de processus et synchronisations. Theoretical Comput. Sci. 30(1), 91–131 (1984)
14. Baeten, J.: A brief history of process algebra. Theoretical Comput. Sci. 335(2–3), 131–146 (2005)
15. Baeten, J., de Vink, E.: Axiomatizing GSOS with termination. In: Alt, H., Ferreira, A. (eds.) STACS 2002. LNCS, vol. 2285, pp. 583–595. Springer, Heidelberg (2002)
16. Baeten, J.C.M., Klop, J.W. (eds.): CONCUR 1990. LNCS, vol. 458. Springer, Heidelberg (1990)
17. Baeten, J., Middelburg, C.A.: Process Algebra with Timing. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin (2002)
18. Baeten, J., Weijland, P.: Process Algebra. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, Cambridge (1990)
19. Bergstra, J., Klop, J.W.: Fixed point semantics in process algebras. Report IW 206, Mathematisch Centrum, Amsterdam (1982)
20. Bergstra, J., Klop, J.W.: Process algebra for synchronous communication. Information and Control 60(1/3), 109–137 (1984)
21. Bloom, S.L., Ésik, Z.: Nonfinite axiomatizability of shuffle inequalities. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 318–333. Springer, Heidelberg (1995)
22. Bloom, S.L., Ésik, Z.: Free shuffle algebras in language varieties. Theoret. Comput. Sci. 163(1-2), 55–98 (1996)
23. Bloom, S.L., Ésik, Z.: Axiomatizing shuffle and concatenation in languages. Inform. and Comput. 139(1), 62–91 (1997)
24. Bloom, S.L., Ésik, Z.: Varieties generated by languages with poset operations. Math. Structures Comput. Sci. 7(6), 701–713 (1997)
25. Bloom, S.L., Ésik, Z.: Shuffle binoids. RAIRO Inform. Théor. Appl. 32(4-6), 175–198 (1998)
26. Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.): ICALP 2006. LNCS, vol. 4052, pp. 10–14. Springer, Heidelberg (2006)

27. de Simone, R.: Higher-level synchronising devices in Meije–SCCS. Theoretical Comput. Sci. 37, 245–267 (1985)
28. Ésik, Z., Bertol, M.: Nonfinite axiomatizability of the equational theory of shuffle. Acta Inform. 35(6), 505–539 (1998)
29. Ésik, Z., Okawa, S.: Series and parallel operations on pomsets. In: Pandu Rangan, C., Raman, V., Ramanujam, R. (eds.) Foundations of Software Technology and Theoretical Computer Science. LNCS, vol. 1738, pp. 316–328. Springer, Heidelberg (1999)
30. Fokkink, W.: Introduction to Process Algebra. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin (2000)
31. Fokkink, W., Luttik, B.: An omega-complete equational specification of interleaving. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 729–743. Springer, Heidelberg (2000)
32. Gischer, J.L.: The equational theory of pomsets. Theoretical Comput. Sci. 61, 199–224 (1988)
33. van Glabbeek, R., Vaandrager, F.: Petri net models for algebraic theories of concurrency. In: de Bakker, J.W., Nijman, A.J., Treleaven, P.C. (eds.) PARLE Parallel Architectures and Languages Europe. LNCS, vol. 259, pp. 224–242. Springer, Heidelberg (1987)
34. Grabowski, J.: On partial languages. Fundamenta Informaticae IV(2), 427–498 (1981)
35. Groote, J.F.: A new strategy for proving ω–completeness with applications in process algebra. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 314–331. Springer, Heidelberg (1990)
36. Hennessy, M.: On the relationship between time and interleaving. Preprint, CMA, Centre de Mathématiques Appliquées, Ecole des Mines de Paris (1981)
37. Hennessy, M.: Algebraic Theory of Processes. MIT Press, Cambridge (1988)
38. Hennessy, M.: Axiomatising finite concurrent processes. SIAM J. Comput. 17(5), 997–1017 (1988)
39. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. J. ACM 32(1), 137–161 (1985)
40. Hoare, C.: Communicating Sequential Processes. Prentice-Hall International, Englewood Cliffs (1985)
41. Jeffrey, A., Schneider, S., Vaandrager, F.: A comparison of additivity axioms in timed transition systems. Report CS-R9366, CWI, Amsterdam (1993)
42. Keller, R.: Formal verification of parallel programs. Commun. ACM 19(7), 371–384 (1976)
43. Milner, R.: Flowgraphs and flow algebras. J. ACM 26(4), 794–818 (1979)
44. Milner, R.: Communication and Concurrency. Prentice-Hall International, Englewood Cliffs (1989)
45. Moller, F.: Axioms for Concurrency. PhD thesis, Department of Computer Science, University of Edinburgh, Report CST-59-89. Also published as ECS-LFCS-89-84 (July 1989)
46. Moller, F.: The importance of the left merge operator in process algebras. In: Paterson, M.S. (ed.) Automata, Languages and Programming. LNCS, vol. 443, pp. 752–764. Springer, Heidelberg (1990)
47. Moller, F.: The nonexistence of finite axiomatisations for CCS congruences. In: Proceedings 5^{th} Annual Symposium on Logic in Computer Science, Philadelphia, USA, pp. 142–153. IEEE Computer Society Press, Los Alamitos (1990)

48. Park, D.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) Theoretical Computer Science. LNCS, vol. 104, pp. 167–183. Springer, Heidelberg (1981)
49. Plotkin, G.D.: A structural approach to operational semantics. Journal of Logic and Algebraic Programming 60–61, 17–139 (2004)
50. Pratt, V.: Modeling concurrency with partial orders. International Journal of Parallel Programming 15(1), 33–71 (1986)
51. Tschantz, S.T.: Languages under concatenation and shuffling. Mathematical Structures in Computer Science 4(4), 505–511 (1994)
52. Yi, W.: Real-time behaviour of asynchronous agents. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 502–520. Springer, Heidelberg (1990)
53. Yi, W.: A Calculus of Real Time Systems. PhD thesis, Chalmers University of Technology, Göteborg, Sweden (1991)

Rule-Based Modelling of Cellular Signalling

Vincent Danos[1,3,4], Jérôme Feret[2], Walter Fontana[3], Russell Harmer[3,4], and Jean Krivine[5]

[1] Plectix Biosystems
[2] École Normale Supérieure
[3] Harvard Medical School
[4] CNRS, Université Denis Diderot
[5] École Polytechnique

Abstract. Modelling is becoming a necessity in studying biological signalling pathways, because the combinatorial complexity of such systems rapidly overwhelms intuitive and qualitative forms of reasoning. Yet, this same combinatorial explosion makes the traditional modelling paradigm based on systems of differential equations impractical. In contrast, agent-based or concurrent languages, such as κ [1,2,3] or the closely related BioNetGen language [4,5,6,7,8,9,10], describe biological interactions in terms of rules, thereby avoiding the combinatorial explosion besetting differential equations. Rules are expressed in an intuitive graphical form that transparently represents biological knowledge. In this way, rules become a natural unit of model building, modification, and discussion. We illustrate this with a sizeable example obtained from refactoring two models of EGF receptor signalling that are based on differential equations [11,12]. An exciting aspect of the agent-based approach is that it naturally lends itself to the identification and analysis of the causal structures that deeply shape the dynamical, and perhaps even evolutionary, characteristics of complex distributed biological systems. In particular, one can adapt the notions of causality and conflict, familiar from concurrency theory, to κ, our representation language of choice. Using the EGF receptor model as an example, we show how causality enables the formalization of the colloquial concept of pathway and, perhaps more surprisingly, how conflict can be used to dissect the signalling dynamics to obtain a qualitative handle on the range of system behaviours. By taming the combinatorial explosion, and exposing the causal structures and key kinetic junctures in a model, agent- and rule-based representations hold promise for making modelling more powerful, more perspicuous, and of appeal to a wider audience.

1 Background

A large majority of models aimed at investigating the behavior of biological pathways are cast in terms of systems of differential equations [11,12,13,14,15,16]. The choice seems natural. The theory of dynamical systems offers an extensive repertoire of mathematical techniques for reasoning about such networks. It provides, at least in the limit of long times, a well-understood ontology of

L. Caires and V.T. Vasconcelos (Eds.): CONCUR 2007, LNCS 4703, pp. 17–41, 2007.

behaviors, like steady states, oscillations, and chaos, along with their linear stability properties. The ready availability of numerical procedures for integrating systems of equations, while varying over parameters and initial conditions, completes a powerful workbench that has successfully carried much of physics and chemical kinetics. Yet, this workbench is showing clear signs of cracking under the ponderous combinatorial complexity of molecular signalling processes, which involve proteins that interact through multiple post-translational modifications and physical associations within an intricate topology of locales [17].

Representations of chemical reaction networks in terms of differential equations are about chemical kinetics, not the unfolding of chemistry. In fact, all molecular species made possible by a set of chemical transformations must be explicitly known in advance for setting up the corresponding system of kinetic equations. Every molecular species has its own concentration variable and an equation describing its rate of change as imparted by all reactions that produce or consume that species. These reactions, too, must be known in advance. Many ion channels, kinases, phosphatases, and receptors – to mention just a few – are proteins that possess multiple sites at which they can be modified by phosphorylation, ubiquitination, methylation, glycosidilation, and a plethora of other chemical tagging processes. About one in a hundred proteins have at least 8 modifiable sites, which means 256 states. A simple heterodimer of two distinct proteins, each with that much state, would weigh in at more than 65,000 equations. It is easily seen that this combinatorics can rapidly amount to more possible chemical species than can be realized by the actual number of molecules involved in a cellular process of this kind. The problem is not so much that a deterministic description is no longer warranted, but rather that the equations, whether deterministic or stochastic, can no longer be written down—and if they could, what would one learn from them?

This difficulty is well recognized. One way out is to use aggregate variables describing sets of modification forms. For example, one might bundle together all phosphoforms of a receptor, regardless of which sites are phosphorylated. This, however, is problematic. First, the choice of what to aggregate and not is unprincipled. Second, the appropriate level of aggregation may change over time as the system dynamics unfolds. Third, the aggregation is error prone, since it has to be done without a prior microscopic description. A further, more subtle, difficulty is that an extensional system of differential equations describes the constituent molecules only in terms of interactions that are relevant in a given context of other molecules. It does not characterize molecular components in terms of their potential interactions that could become relevant if the composition of the system were to change. As a consequence, "compositional perturbations", such as adding a novel molecular component (a drug) or modifying an extant one (to model the effects of knocking out a site or adding a new domain) are virtually impossible to carry out by hand, since they require, again, enumerating all chemical consequences in advance and then rewriting all affected equations.

These problems have led to recent attempts at describing molecular reaction networks in terms of molecules as "agents", whose possible interactions are defined by rules that specify how a local pattern of "sites" and their "states" is to be rewritten [18,19]. This resembles good old organic chemistry, except that biologists think of post-translational modifications less as chemical transformations (which, of course, they ultimately are) than as state changes of the *same* agent. A phosphorylated kinase is, at some useful level of description, still the same entity - though in a different state - than its unphosphorylated version. Indeed, biologists think of the state of an agent as a specific set of interaction capabilities. The discontinuous change of such capabilities despite an underlying continuity in agent-type hinges on the large size of a protein, which allows for a significant change in hydrophobic and electrostatic dispositions without much changing the protein's overall chemical identity.

A rule may specify, for example, that if site Y996 of protein A is phosphorylated, protein B can bind to it with its site SH2. Since this rule applies regardless of whether A or B are bound to other partners or possess other sites with particular states, it captures a potentially large set of individual reactions between distinct molecular species. The need for spelling out all these reactions was spelling problems for "flat" (extensional) reaction network representations, whereas modifying or extending a reaction system is now as simple as modifying a single rule or merging sets of rules, respectively.

Our stance in this paper is to forgo writing out differential equations, and directly operate at the level of rules defining the interactions among a set of agents. Biological signalling and control processes are, in fact, massively distributed systems, and this has led Regev et al. to propose Milner's π-calculus [20], a minimal language for describing concurrent systems, as a language for modelling biological systems [21,22,23]. Since then, numerous variants of representations emphasizing different types of biological processes have been put forward [19,24,25,26,27]. We shall use the language κ [2,3], as a direct and transparent formalisation of molecular agents and their interactions in signalling networks. Most of the points we shall advance here are, however, independent of any committment to a particular syntax, as long as domain-level modifications and bindings can be represented, and one can condition those on the binding and internal states of the various entities participating to a reaction.

Taking concurrency seriously means understanding the organization of such systems in terms of observables defined from within rather than outside these systems. Time is a particular case in point. In molecular systems, the temporal precedence among events cannot be defined (at first) on physical time, since cells or molecules do not bear watches, let alone synchronized ones. It is well known in concurrency that temporal precedence is a logical relation that gives rise to a partial order, as opposed to a total order. Some events must occur before others can happen, while other events may happen in any sequence, reflecting their mutual independence. Clearly, in any particular physical realization one will observe a particular sequence of events. The issue, however, is to uncover which aspects of that sequence are necessary and which contingent. The issue is to discover the invariant structure underlying all observable sequences. Differential equations are unable to resolve this causality, precisely because they treat time as global, as if everything proceeded in a synchronized fashion.

In contrast, concurrency approaches have long sought to understand the dependencies that constrain an observable event. The traditional notions of causality developed in concurrent models (such as Petri nets, a language which is equivalent to structure-less reactions [28,29]) can be adapted to κ, and used to clarify how a path toward a specified event has unfolded from initial conditions. It is worth mentioning that κ can be seen as an applied graph-rewriting framework, and as such, belongs to a family of formalisms where the notions of causality, conflict, and the attendent concept of event structures, are well-understood [30]. Similar notions of causality have been derived for π-calculus itself, and some have been put to use in a bio-modelling scenario with an ambition to decribe the inner workings of a pathway by deriving causal traces (aka minimal configurations in the event structure terminology) [31,32]. What we shall use here is a related but subtler notion of *minimal* causal path, or *story*, which seems an appropriate formalization of what biologists colloquially call a "signalling pathway", and may have an interest which is independent of the intended application. Used in conjunction with stochastic simulation, stories can generate insights into the collective properties of rule-based systems. We will show how this works using an EGF receptor signalling model that would be quite large and unwieldy by traditional standards. On the basis of relationships of inhibition (or conflict) between rules, we will also see that one can identify key junctures in the system's dynamics that yield explanatory insights and suggest numerical experiments.

To introduce the framework, we warm up with a simple example of an ubiquitous control motif in cellular signal processing: a futile cycle of enzymatic modification and demodification of a target substrate. In general, we have set for an easy and accessible style of explanation, where definitions are precise but not formal. The interested reader will find it easy to reconstruct the technical underpinnings, as we have given a measure of further technical details in an appendix. It may also be worth mentioning that the notions presented here have been implemented, and in both the short preliminary example and the larger EGF receptor one, we have used those implementations to obtain the various simulations, causal traces, and stories.

2 A Futile Cycle

2.1 Agents and Rules

The κ description of a system consists of a collection of *agents* and *rules*. An agent has a name and a number of labeled sites, collectively referred to as the agent's interface. A site may have an internal state, typically used to denote its phosphorylation status or other post-translational modification. Rules provide a concise description of how agents interact. Elementary interactions consist of the binding or unbinding of two agents, the modification of the state of a site, and the deletion or creation of an agent. This seems limited, but closely matches the style of reasoning that molecular biologists apply to mechanistic interactions in cellular signalling. While this approach does not address all aspects of signaling (such as compartmentation), it does cover a substantive body of events sufficient for the present purpose.

To develop the main concepts, we start with a system consisting of three agents: a kinase K, a target T with two phosphorylatable sites x and y, and a phosphatase P. We first describe a phosphorylation event by means of three elementary actions and their corresponding rules: (1) the kinase K binds its target T either at site x or y; (2) the kinase may (but need not) phosphorylate the site to which it is bound; (3) the kinase dissociates (unbinds) from its target. For ease of reference, we label rules with a mnemonic on the left. Using a textual notation, we represent internal states as '`~u`' (unphosphorylated), and '`~p`' (phosphorylated), and physical associations (bindings or links) as '`!`' with shared indices across agents to indicate the two endpoints of a link. The left hand side of a rule specifies a condition in the form of a pattern expressed as a partial graph, which represents binding states and site values of agents. The right hand side of a rule specifies (usually elementary) changes to agents mentioned on the left. A double arrow indicates a reversible rule, the name refers to the forward version of the rule say `r`, while the opposite rule is written `r_op`. With these conventions, the phosphorylation process of sites x or y translates into:

```
'KT@x' K(a),T(x) <-> K(a!1),T(x!1)
'Tp@x' K(a!1),T(x~u!1) -> K(a!1),T(x~p!1)
'KT@y' K(a),T(y) <-> K(a!1),T(y!1)
'Tp@y' K(a!1),T(y~u!1) -> K(a!1),T(y~p!1)
```

Note that not all sites of an agent interface need to be present in a rule, eg in the first rule `KT@x`, T's interface does not mention y. Likewise, if a site is mentioned at all, its internal state may be left unspecified, eg in the same first rule one does not say whether site x in T is phosphorylated or not. This is the '*don't care, don't write*' convention, only the information which is conditioning the triggering of a rule needs to be represented.

The action of the phosphatase P, which undoes the action of K, is described by a set of similar rules:

```
'PT@x' P(a),T(x) <-> P(a!1),T(x!1)
'Tu@x' P(a!1),T(x~p!1) -> P(a!1),T(x~u!1)
'PT@y' P(a),T(y) <-> P(a!1),T(y!1)
'Tu@y' P(a!1),T(y~p!1) -> P(a!1),T(y~u!1)
```

It is possible to associate rate constants with each rule, as we shall do later. We refer to this rule set as the Goldbeter-Koshland (GK) loop [33]. It is a frequent motif that appears in many variants throughout cellular signal transduction. Notice how the specification of elementary actions forces us to make our mechanistic choices explicit. The example views phosphorylation of T by K as a distributed mechanism, whereby the kinase lets go of its target before phosphorylating it (or another instance) again, since it cannot remain bound to site x and phosphorylate site y. Other variants of multisite phosphorylation involve a processive mechanism whereby the same kinase acts sequentially on some or all sites of its target. Further variants still would have to specify whether multiple kinases can be bound to the same target or not.

2.2 The Contact Map

Large rule sets can be difficult to understand, and it is important to have a suite of views that report at a glance information implied by the rules. Such cognitive devices are useful for a piecewise modular construction of the system of interest. They also allow for a modicum of reasoning about the system. A first useful view of the rule set is the *contact map*, which is akin to a protein-protein interaction (PPI) map and is shown in Fig. 1. The contact map is a graph whose nodes are the agents with their interfaces and whose edges represent possible bindings between sites. Potential site modifications are indicated by a colour code. The contact map does not provide causal information, in the sense that it does not specify which conditions must be met for a modification or binding to occur. It only shows possibilities.

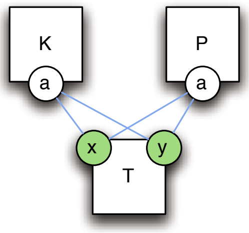


Fig. 1. A Goldbeter-Koshland contact map: nodes represent the three kinds of agents in the rule set, together with their sites, and each site is connected to sites it can bind to; whether a site can be modified is indicated by a colour code (green). Although very simple, the associated rule set generates already 38 non-isomorphic complexes (36 of which contain T).

2.3 Stochastic Simulation

With a rule set in place, one can generate time courses, or stochastic trajectories, for user-defined *observables*. Here we choose an initial state consisting of a 100 of each of the three agents with their interfaces in defined states, `T(x~u,y~u)`, `K(a)`, `P(a)`. We decide to track two observables: (1) the number of doubly phosphorylated target molecules, regardless of whether sites x and y are bound to enzymes, which is written as `T(x~p?,y~p?)`, and (2) the number of target instances T that are fully phosphorylated but free on both sites x and y, `T(x~p,y~p)`. As can be verified in Fig. 2, the latter observable has to be smaller since it is more stringent. The trajectories are obtained using an entirely rule-based version of Gillespie's kinetics which generates a continuous time Markov chain [34]. At any given time a rule can apply to a given state in a number of ways. That number is multiplied by the rate of the rule and defines the rule's *activity* or flux in that state of the system. It determines the likelihood that this rule will fire next, while the total activity of the system determines probabilistically the associated time advance. This simulation principle can be implemented within κ with rather nice complexity properties, since the cost of a simulation event (triggering a rule) can be made independent on the size of the agent population and depends only logarithmically in the number of rules. This is very useful when it comes to larger systems. One thing to keep in mind is that any obtained trajectory is but one realization of a stochastic process that will differ slightly when repeated. Sometimes, but not always, their average behaviour can be captured in a suitable differential system. Also, and evidently, the trajectories will depend on the choice of rates (see the captions to Fig. 2(a) and 2(b)).

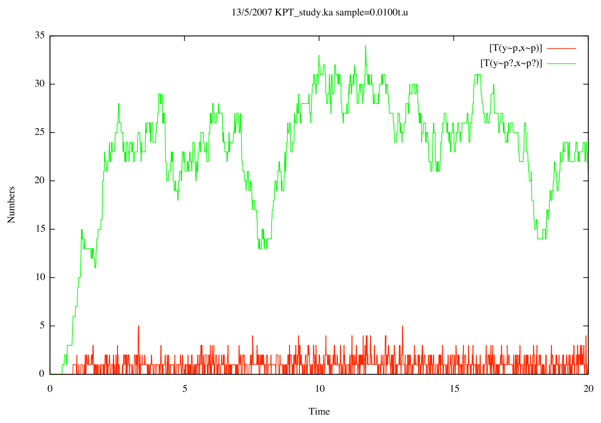
2.4 Precedence and Stories

The trajectory samples obtained above represent an agent-centric view of the system's evolution. For instance, they do not answer directly the question of which succession of events results in a fully phosphorylated form of the target T. This is where the notion of pathway or *story* comes into play. An event, which is the application of a rule, is said to precede a posterior event, if those events don't commute, which can be 1) either because the latter cannot possibly happen before the former, ie has to be after, or 2) because the former can no longer happen after the latter, ie has to be before. Eg an event of type `Tu@x` has to be after an event of type `PT@x`, and before an event of type `PT@x_op`. This notion of logical precedence or causation defines a partial order on any sequence of events along a trajectory. The fact that two events may succeed one another in a particular trajectory does not imply that they are in such a relationship. An example is provided by two successive events of type `KT@x` and `KT@y`. Events that are not related by precedence are said to be concurrent.

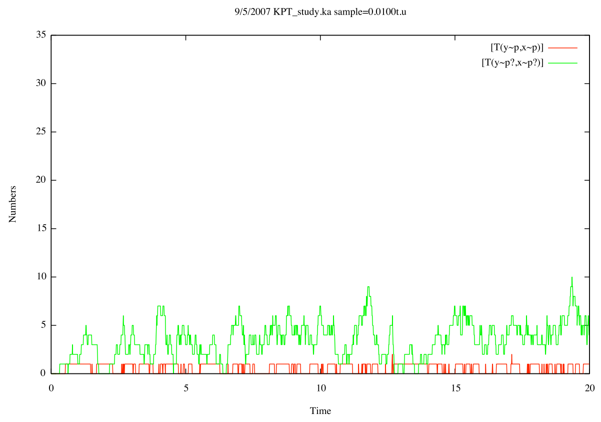
The idea behind a story is to retain only those events in the causal lineage that contributed a net progression towards an event of interest, or in other words a story summarises how a given event type can be obtained. This means in particular that circular histories which generate a situation that is subsequently undone without leaving a side effect that impacts the causal lineage later on should be eliminated. We therefore define a story as a sequence of events that:

- begins with the initial condition and ends with an event of a given type called the observable,
- consists only of events that are in the causal lineage to the observable (which eliminates events that are concurrent to the observable)
- contains no event subsequence with the same properties (which in particular eliminates circles).

Taking as an initial condition one instance of each agent, and as an observable the doubly phosphorylated form of the target, one obtains two stories depending on whether K hits x or y first (Fig. 3 shows the former). If the initial condition were to contain more than one K, there would be a third story in which both sites are phosphorylated by different K-agents.



(a) Association and modification rates are set to 1, and dissociation rates are set to 10 (per time units).



(b) Same model perturbed by a tenfold increase in the association and modification rate of P at site x.

Fig. 2. Goldbeter-Koshland loop simulation: the initial state has a hundred copies of each agent, each disconnected and unphosphorylated; the level of doubly phosphorylated T is lower in the perturbed case (right)

2.5 Inhibition and Activation

Say a rule inhibits another one if the former can destroy an instance of the latter. Note that this may not be a symmetric relationship. An example is given by the application of the rules `KT@x` and `PT@x` (symmetric), or the rules `PT@x_op` and `Tu@x` (dissymmetric). Similarly, say a rule activates another one if the former can create a new instance of the latter. An example is `PT@x` which may create a new instance of `Tu@x`.

Superimposing such inhibitions on a story, as in Fig. 3 above, suggest ways in which one can prevent or delay a story's ending. Indeed, numerically, a tenfold

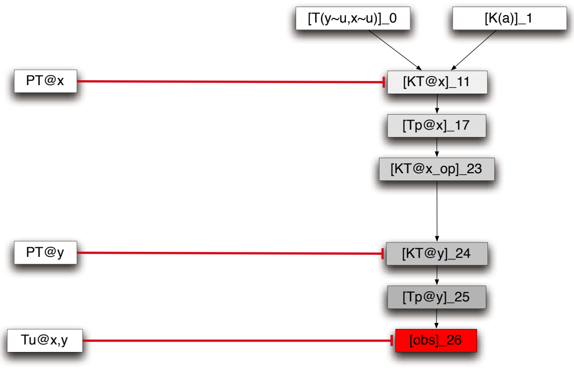


Fig. 3. A story: the story observable is at the bottom, causal depth is represented by shades of grey. Event numbers represent the actual step in the simulation where these events occurred (missing events are either concurrent, or compressible). Various inhibitions attached to the story are shown on the left (explained below).

increase in the rate constants of `PT@x` and `Tu@x` yields a much lower value for the observable (see Fig. 2(b)).

We now proceed to apply these ideas to the more complex example of an EGF receptor model coupled to a MAP kinase cascade. In its full form, as presented in Ref. [35], EGFR signaling is a complex suite of pathways whose boundaries to other signaling systems appear increasingly blurred. While the present model falls far short of representing this complexity, it goes some way towards demonstrating how to eventually represent and face it.

3 The EGFR Model

3.1 EGFR Model Elements

The EGFR signalling network plays an important, yet only partially understood, role in regulating major events in mammalian cells, such as growth, proliferation, survival, and differentiation. In outline, a signal arrives at the cell membrane in the form of a ligand, EGF, which binds to the extra-cellular portion of a special receptor protein, EGFR, that straddles the membrane. With the arrival of EGF, an EGFR becomes capable of binding to a neighbouring EGFR also bound to a ligand. Such receptor pairs can cross-activate one another, meaning that certain of their intra-cellular residues become phosphorylated. These phosphorylated residues now serve as binding sites for a variety of proteins in the cytoplasm. This in turn leads to the activation of a small protein, Ras, that serves as a kind of relay for triggering a cascade of phosphorylations comprising three stacked GK loops in which the fully phosphorylated form of one loop acts as the kinase of the next, causing the overall cascade to behave like an amplifier and culminating in the activation of ERK.

The pathway to the activation of ERK has been the target of an intense modelling effort over the past decade. The ubiquity and importance of the pathway for biomedical applications [36, Chap. 5] have spurred extensive studies at the mechanistic level of protein-protein interactions and localizations. At the same time, the subtleties uncovered by these investigations (see, for example, the receptor network combinatorics of the pathway [37]) have made it clear that intuition alone, however sharp, cannot confront its complexity and only risks flushing enormous amounts of drug development money down the drain. To calibrate the reader on the magnitudes involved, the particular model presented below contains 70 rules and generates over 10^{23} distinct molecular species (see appendix for the complete and commented rule set). An exhaustive approach with differential equations would, in principle, require that many equations—and that is by no means a large example.

Fig. 4 depicts the model's contact map. The associated rule-based model over-

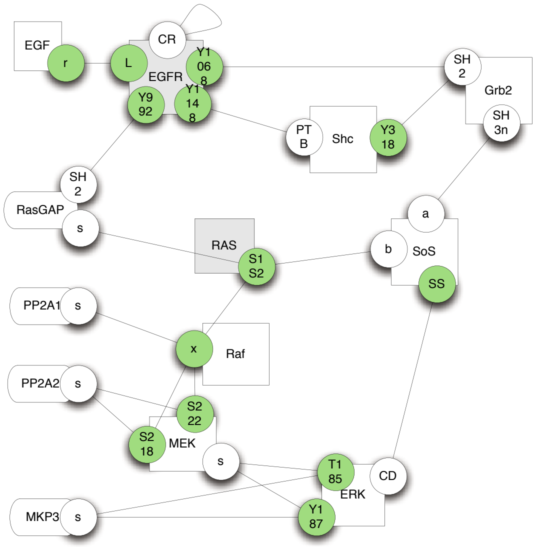


Fig. 4. The contact map of the EGFR/ERK model

lays a causal structure on the contact map by specifying, for example, that Ras can only bind Raf if it has been phosphorylated beforehand at site S1S2. The rule set itself was mainly obtained from refactoring two existing ordinary differential equations (ODE) models [11,12] treating two different aspects of EGF-EGFR signalling: down-regulation of Erk activity through a negative feedback and down-regulation of the signal through internalization.

In its present form, our EGFR model comprises three modules. Firstly, a receptor module, which for illustration purposes retains only the EGFR (or ErbB1) receptor, but includes internalization dynamics using fictitious sites whose state flags localization. This module can be refined to include a receptor heterodimerization network comprising all four receptors of the ErbB family. Secondly, adaptors and relays, such as Sos, Grb2, and Ras. These too can be extended as the complexity of the model is built up in a stepwise fashion. Finally, we have the module containing the target MAPK cascade.

3.2 Ras Activation

An examination of the contact map immediately reveals a potentially critical role for Ras in the behaviour of the model: Ras has only one site but can bind to three distinct agents (SoS, RasGAP and Raf) and so probably forms a bottleneck. Inspection of the rules governing these four agents allows us to refine the contact map: Ras's site has an internal state (representing active or inactive), only SoS can bind to an inactive Ras, whereas RasGAP and Raf must compete for active Ras.

In terms of signal propagation, SoS activates Ras so that Ras can in turn activate Raf. RasGAP plays an inhibitory role by deactivating Ras. We can observe this chain of causality by looking at the two stories leading to Ras's activation (Fig. 5 and 6, events are labeled by rule names as given in the appendix). Each of the two stories contains a rule that inhibits the other one, respectively `Shc_Grb2` and `EGFR_Grb2`, a clue that these stories are competing ways to obtain the observable. In the former, Grb2 binds to the receptor directly, in the latter it binds to Shc. Only the former would resist a Shc knock-out.

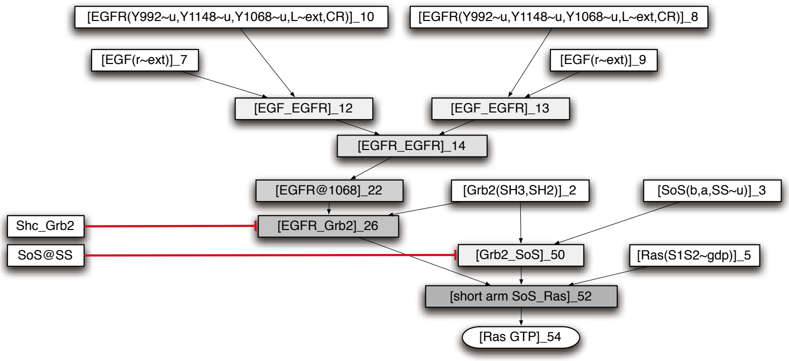


Fig. 5. Short arm activation of Ras (without Shc); inhibitions by `SoS@SS` and `Shc_Grb2` shown on the left; the observable is the activation rule `Ras GTP` (oval shape)

RasGAP does not appear in either of the above stories, confirming that it plays no role in the logical propagation of the signal. RasGAP, however, does play a role in shaping the kinetics of signal propagation. Indeed, most sequences of events leading to Raf's recruitment do exhibit RasGAP intervention (and are therefore not stories). This slightly paradoxical effect of the EGF signal is neatly captured by the *negative feed-forward loop* in Fig. 7 (negative because it has a negative effect on the story, via the rule `direct RasGAP_Ras`, forward because that effect proceeds from a prior event to the story end). In order to propagate the signal, SoS is induced to activate Ras (and hence the downstream cascade to ERK) but, at the

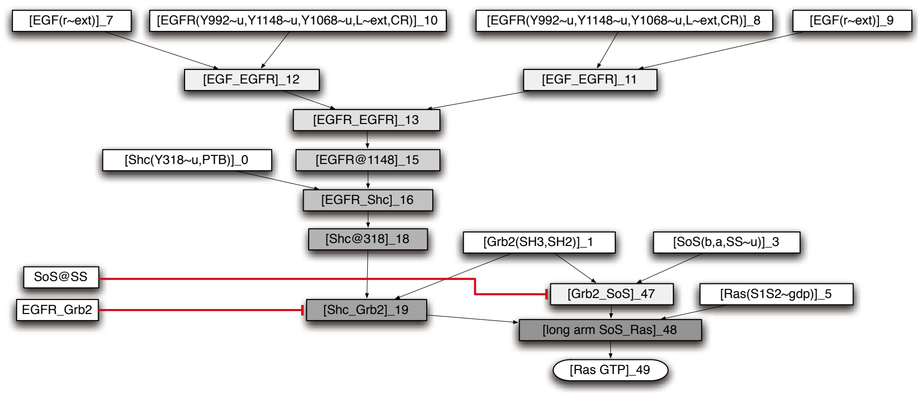


Fig. 6. Long arm activation of Ras (with Shc); inhibitions by `SoS@SS` and `EGFR_Grb2` shown on the left

same time, the signal also induces RasGAP to frustrate SoS's work. Of course, a signal needs to be controlled and eventually down-regulated, so the existence of a RasGAP-like agent should be expected. Both the positive (SoS) and the negative (RasGAP) influences on Ras depend on the same signal which suggests that Ras's activation dynamics only depend on the relative concentrations of SoS and RasGAP: if RasGAP dominates, Ras activation will be weak and short-lived; if SoS dominates, it will be stronger and last longer.

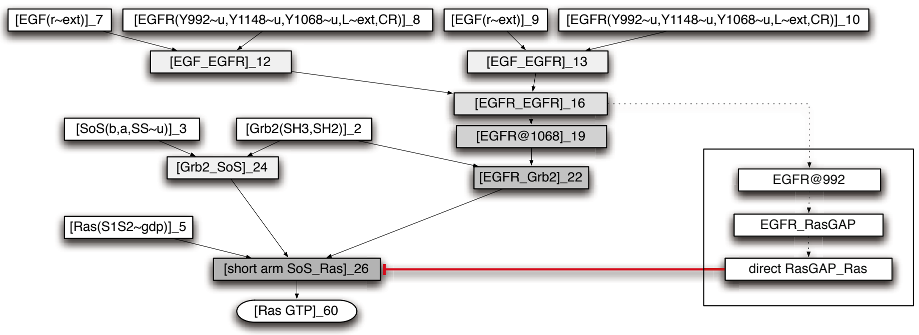


Fig. 7. Battle between SoS and RasGAP; negative feed-forward loop from EGFR dimerisation to Ras activation shown on the right (dotted arrows are activations, the blunted arrow is an inhibition)

3.3 SoS Deactivation

As can be seen in Fig. 5, SoS has a second enemy in the form of activated ERK which by rule `SoS@SS` may inhibit the formation of the complex between Grb2 and SoS by phosphorylating SoS. Fig. 8 shows one of the two stories leading to activated ERK (the other uses the long arm), and there appears a *negative feedback loop*, where the end inhibits an event internal to its own story.

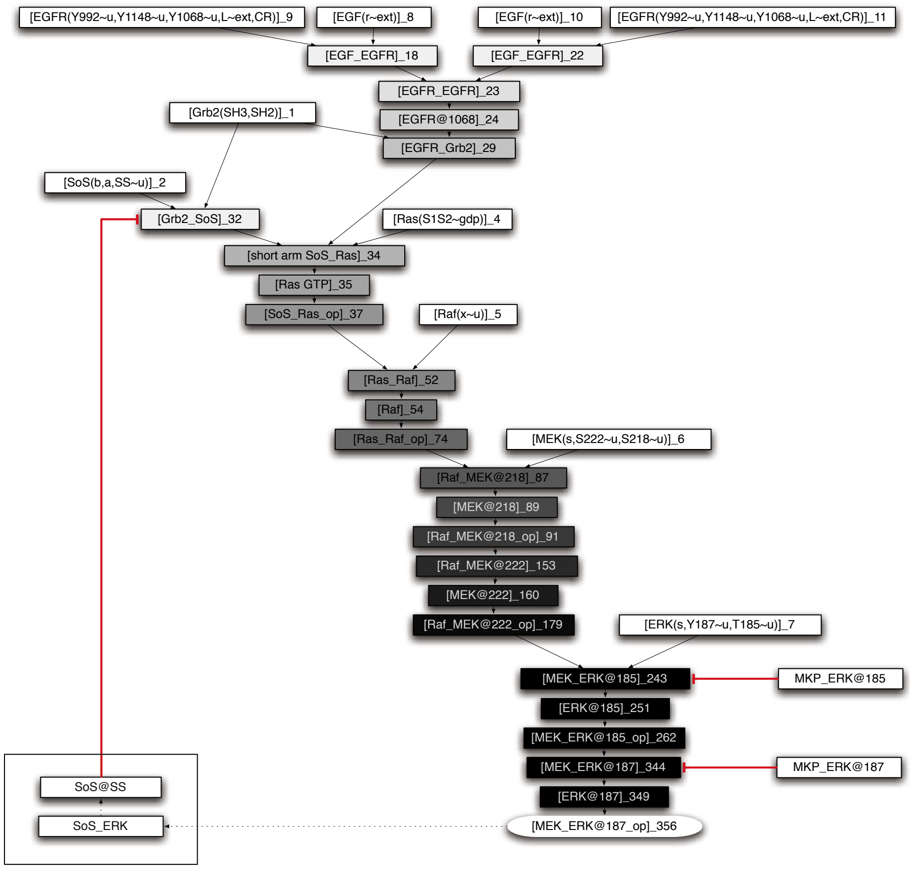
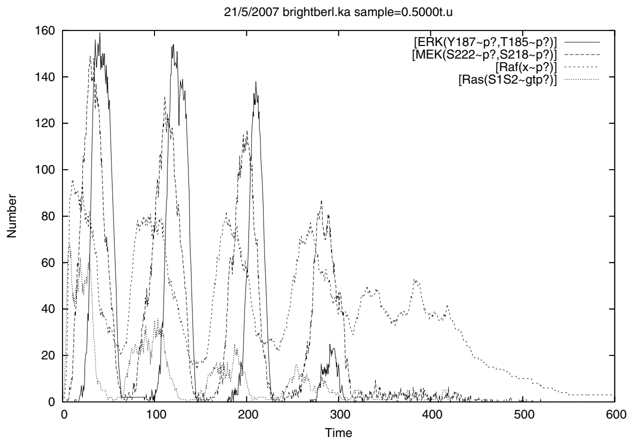


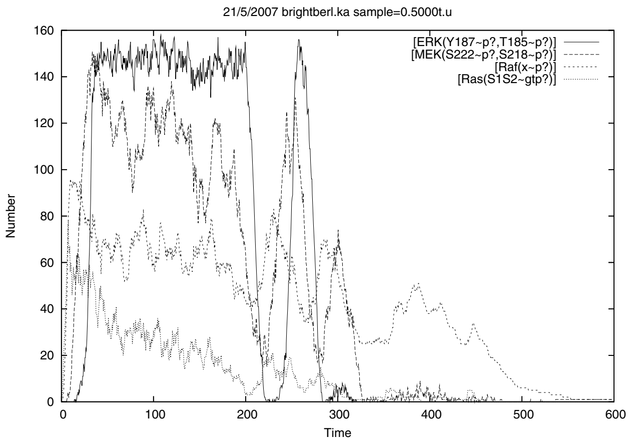
Fig. 8. Short arm story to ERK activation; negative feedback look from active ERK to SoS shown on the left; MKP3 inhibitions shown on the right

For the duration of the SoS's phosphorylation, this substantially weakens the signal from SoS to Ras, essentially shifting the balance in favour of RasGAP. As a result, the level of active Ras decreases which, with a small delay, causes a significant reduction in active ERK at the bottom of the cascade. At that moment, SoS is no longer being strongly targeted by ERK and can, once more, signal to Ras. We thus expect a cyclic process of activation and then inhibition of Ras, leading to an oscillating activation pattern for ERK, typical of a cascade embedded in a negative feedback loop [16].

A crucial parameter determining the shape of these oscillations is the "recovery rate" of SoS from its phosphorylation by ERK. A slow recovery rate leads to a clear oscillation of the cascade. As the rate of recovery increases, the cascade oscillates more quickly, albeit with the same amplitude. With a sufficiently rapid recovery rate, the cascades achieves a transient activation, again with the same amplitude, with a little oscillation as the signal dies (Fig. 9). Thus the qualitative observation embodied in Fig. 8 is rather well echoed at the quantitative level.



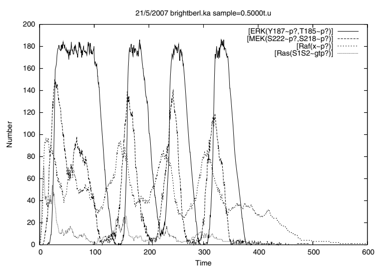
(a) Slow recovery rate of SoS.



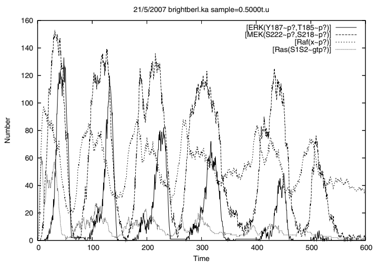
(b) Fast recovery rate for SoS.

Fig. 9. Oscillations and rates of SoS recovery

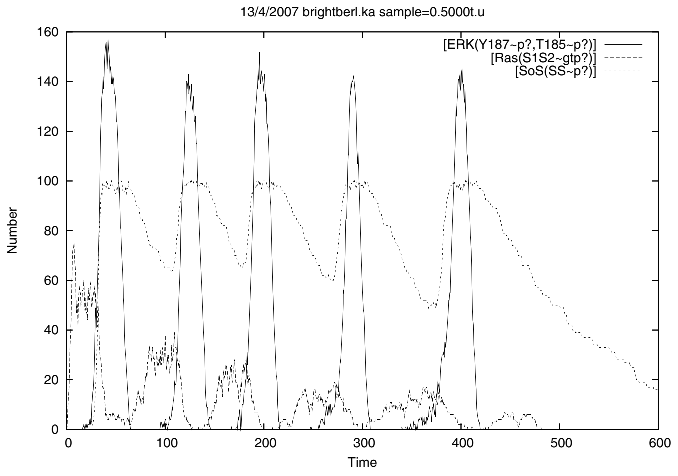
Another factor regulating the effect of the negative feedback to SoS comes from MKP3, the phosphatase targeting ERK, as suggested by the two inhibitions shown on the right of the activated ERK story (Fig. 8). A higher concentration of MKP3 will tend to hamper activation of ERK and this impacts the rate of oscillation at the cascade: increasing the concentration of MKP3 over successive simulations, one observes (without surprise) that the amplitude of ERK activation decreases. However, one also observes that ERK remains active for less time, leading to a gradual increase in the frequency of oscillation (Fig. 10). In addition, the signal attenuates faster: with more phosphatase in the system, ERK requires a higher "threshold" concentration of its kinase (MEK) in order to achieve significant activation. While this obviously goes against ERK activation, it also protects SoS from being in turn inhibited by ERK. However, this



(a) Low concentration of MKP3.



(b) High concentration of MKP3



(c) Combination of fast feedback and slow recovery

Fig. 10. Impact of MKP3 concentration on pathway oscillations

secondary effect turns out to be fairly minor, since a typical system contains more ERK than SoS molecules. Therefore, even a reduced level of ERK activation suffices to mount a powerful inhibition of SoS.

One final parameter substantially influences ERK's level of activation: the speed of SoS phosphorylation by ERK (ie the strength of the `SoS@SS` inhibition shown Fig. 5 and 6). A slow rate limits the effect of the negative feedback, leading to a longer and steadier period of Ras and ERK activation and little to no oscillation. A fast rate accentuates the negative feedback effect, considerably shortening the time during which Ras signals and, in tandem with a slow recovery rate, leads to a pronounced, low-frequency oscillation (Fig. 10).

4 Conclusions

We have illustrated how rule-based modeling transcends the bottleneck of the traditional ODE-based framework. It does so in many important ways, both practical and conceptual, which we summarize here. First, and this is the starting point, rule-based modeling tames combinatorial explosion by decontextualizing reactions between molecular species into rules defined on patterns. As an immediate corollary, modifications and extensions become fairly straightforward.

Rules represent nuggets of mechanistic knowledge that current experimental practice is rapidly accumulating. Rather than expressing such knowledge in terms of human language or non-executable graphical information, it seems vastly more useful to represent it in a context-free grammar ready for computational consumption. Much like chemical reactions, rules can be viewed as operational "instructions" that can be let loose on a set of molecular agents, driving the unfolding of pathways and their kinetics. In this sense, κ-rules make knowledge executable. The granularity of such rules is, in principle, adaptable to the needs of lab scientists. We believe that the current level of granularity offered by κ or the variant language BNG [18] meets the most urgent practical needs.

Sets of rules are not just inputs to simulators, like systems of differential equations are not just inputs to numerical integrators. Rather, rule sets replace systems of differential equations as formal entities that can be subject to rigorous analysis from which to extract predictive and explanatory information about the behavior of systems. In contrast to the synchronicity of differential equations, rules operate in a concurrent execution model, which is a far more appropriate representation of what is actually going on in cells. This constitutes rather unfamiliar terrain for many biologists, yet this is a turf that has been successfully plowed for over thirty years in computer science. We have shown how notions of conflict and causation can be used to build maps that relate rules to one another. We have defined a concept of story which we believe formalizes the intuitive notion of "pathway" that biologists entertain. Stories are partial orders representing causal lineages that explain how a given observable arises in logical time. Stories change over time, since they depend on available molecular resources. The superposition of stories with rule inhibition maps identifies potential "story spoilers", junctures at which logical structure meets kinetics. We have made extensive use of this trick to explain several dynamical and logical features of a simplified EGFR signalling system. Our experience is that this mode of reasoning matches quite naturally the way biologists intuitively go about telling their "stories". The only difference is that our framework formalizes this process and therefore enables computational procedures to tackle much more complicated systems in rigorous ways when the story-telling of biologists risks degenerating into just that.

It is worth emphasizing that the main dynamic characteristics of our modest EGFR case were obtained with uniform rate constants for all rules. The impact of certain rules on these characteristics was then explored by varying certain rate constants. This is not to say that rate constants don't matter, but it does hint at the importance of the causal architecture of a system in shaping dynamics. Disentangling the contribution of causal structure and rates to overall systems dynamics is hardly possible in large systems of differential equations.

By forgoing this separation, modelers who fit rate constants to ODE systems risk engaging in an idle encoding exercise rather than a modeling process, since many behaviors can be inscribed into any sufficiently large system of ODEs by appropriate choice of rate parameters. We believe that rule-based modeling affords a strategy whereby one first tries to get the logical structure to generate key dynamical characteristics and then tunes the rate constants to obtain the fine structure. If the logical structure is insufficient, the odds are that our knowledge is insufficient and that more experiments would be better than more rate tuning.

It is useful to think of rule-based modeling as a task in concurrent programming, where rules are computational instructions that contribute to system behavior, as in the bigraphical reactive systems [38]. It is difficult to grasp how concurrent systems – in particular natural ones like cells, tissues, and organisms – function and why they function the way they do. Modeling in a rule-based format yields a better appreciation of the role played by individual mechanisms in generating collective behavior. Linking architecture to behavior will produce more informed strategies for intervention in the case of disease and will help us distill the principles that enabled cells to evolve such versatile information processing systems in the first place. As in programming, however, there is ample opportunity for mistakes. In fact, a model might be wrong not because it isn't a correct description of the world, but because it may not express what the modeler intended (think typo). To catch such mistakes is crucial and will eventually necessitate a veritable "modeling environment" with sophisticated debugging and verification tools. The complete absence of such tools in the traditional ODE framework, makes classic models beyond a certain size highly prone to error and exceedingly difficult to maintain. As in programming, rule-based models are "grown" by merging smaller models to build larger models that are easily refined by incorporating new empirical knowledge. Rule-based modeling is as much a scientific instrument as it is effective knowledge management.

References

1. Curien, P.L., Danos, V., Krivine, J., Zhang, M.: Computational self-assembly (submitted) (February 2007)
2. Danos, V., Laneve, C.: Formal molecular biology. Theoretical Computer Science 325(1), 69–110 (2004)
3. Danos, V., Laneve, C.: Core formal molecular biology. In: Degano, P. (ed.) ESOP 2003 and ETAPS 2003. LNCS, vol. 2618, pp. 302–318. Springer, Heidelberg (2003)
4. Faeder, J., Blinov, M.B.G., Hlavacek, W.: BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. Complexity 10, 22–41 (2005)
5. Blinov, M., Yang, J., Faeder, J., Hlavacek, W.: Graph theory for rule-based modeling of biochemical networks. In: Proc. BioCONCUR 2005 (2006)
6. Faeder, J., Blinov, M., Hlavacek, W.: Graphical rule-based representation of signal-transduction networks. In: Proc. ACM Symp. Appl. Computing, pp. 133–140. ACM Press, New York (2005)

7. Faeder, J.R., Blinov, M.L., Goldstein, B., Hlavacek, W.S.: Combinatorial complexity and dynamical restriction of network flows in signal transduction. Systems Biology 2(1), 5–15 (2005)
8. Blinov, M.L., Yang, J., Faeder, J.R., Hlavacek, W.S.: Depicting signaling cascades. Nat. Biotechnol. 24(2), 1–2 (2006)
9. Blinov, M.L., Faeder, J.R., Goldstein, B., Hlavacek, W.S.: A network model of early events in epidermal growth factor receptor signaling that accounts for combinatorial complexity. BioSystems 83, 136–151 (2006)
10. Hlavacek, W., Faeder, J., Blinov, M., Posner, R., Hucka, M., Fontana, W.: Rules for Modeling Signal-Transduction Systems. Science's STKE 2006. 344 (2006)
11. Brightman, F., Fell, D.: Differential feedback regulation of the MAPK cascade underlies the quantitative differences in EGF and NGF signalling in PC12 cells. FEBS Lett. 482(3), 169–174 (2000)
12. Schoeberl, B., Eichler-Jonsson, C., Gilles, E.D., Müller, G.: Computational modeling of the dynamics of the map kinase cascade activated by surface and internalized EGF receptors. Nature Biotechnology 20, 370–375 (2002)
13. Orton, R.J., Sturm, O.E., Vyshemirsky, V., Calder, M., Gilbert, D.R., Kolch, W.: Computational modelling of the receptor tyrosine kinase activated MAPK pathway. Biochemical Journal 392(2), 249–261 (2005)
14. Huang, C., Ferrell, J.: Ultrasensitivity in the mitogen-activated protein kinase cascade (1996)
15. Kholodenko, B., Demin, O., Moehren, G., Hoek, J.: Quantification of Short Term Signaling by the Epidermal Growth Factor Receptor. Journal of Biological Chemistry 274(42), 30169–30181 (1999)
16. Kholodenko, B.: Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades (2000)
17. Pawson, T., Nash, P.: Assembly of Cell Regulatory Systems Through Protein Interaction Domains. Science 300(5618), 445–452 (2003)
18. Blinov, M., Faeder, J., Hlavacek, W.: BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. Bioinformatics 20, 3289–3292 (2004)
19. Eker, S., Knapp, M., Laderoute, K., Lincoln, P., Meseguer, J., Sonmez, K.: Pathway logic: Symbolic analysis of biological signaling. In: Proceedings of the Pacific Symposium on Biocomputing. pp. 400–412 (January 2002)
20. Milner, R.: Communicating and mobile systems: the π-calculus. Cambridge University Press, Cambridge (1999)
21. Regev, A., Silverman, W., Shapiro, E.: Representation and simulation of biochemical processes using the π-calculus process algebra. In: Altman, R.B., Dunker, A.K., Hunter, L., Klein, T.E. (eds.) Pacific Symposium on Biocomputing, vol. 6, pp. 459–470. World Scientific Press, Singapore (2001)
22. Priami, C., Regev, A., Shapiro, E., Silverman, W.: Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Information Processing Letters (2001)
23. Regev, A., Shapiro, E.: Cells as computation. Nature 419 (September 2002)
24. Regev, A., Panina, E.M., Silverman, W., Cardelli, L., Shapiro, E.: Bioambients: An abstraction for biological compartments. Theoretical Computer Science (2003) (to appear)
25. Cardelli, L.: Brane calculi. In: Proceedings of BIO-CONCUR'03, Marseille, France. Electronic Notes in Theoretical Computer Science, Elsevier, Amsterdam (2003) (to appear)

26. Priami, C., Quaglia, P.: Beta binders for biological interactions. Proceedings of CMSB 3082, 20–33 (2004)
27. Danos, V., Krivine, J.: Formal molecular biology done in CCS. In: Proceedings of BIO-CONCUR'03, Marseille, France. Electronic Notes in Theoretical Computer Science, Elsevier, Amsterdam (2003) (to appear)
28. Nielsen, M., Winskel, G.: Models For Concurrency. In: Handbook of Logic and the Foundations of Computer Science, vol. 4, pp. 1–148. Oxford University Press, Oxford (1995)
29. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains. Theoretical Computer Science 13, 85–108 (1981)
30. Baldan, P., Corradini, A., Montanari, U.: Unfolding and event structure semantics for graph grammars. In: Thomas, W. (ed.) ETAPS 1999 and FOSSACS 1999. LNCS, vol. 1578, pp. 367–386. Springer, Heidelberg (1999)
31. Baldi, C., Degano, P., Priami, C.: Causal pi-calculus for biochemical modeling. In: Proceedings of the AI*IA Workshop on BioInformatics 2002, pp. 69–72 (2002)
32. Curti, M., Degano, P., Priami, C., Baldari, C.: Modelling biochemical pathways through enhanced–calculus. Theoretical Computer Science 325(1), 111–140 (2004)
33. Goldbeter, A., Koshland, D.: An Amplified Sensitivity Arising from Covalent Modification in Biological Systems. Proceedings of the National Academy of Sciences 78(11), 6840–6844 (1981)
34. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem 81, 2340–2361 (1977)
35. Oda, K., Matsuoka, Y., Funahashi, A., Kitano, H.: A comprehensive pathway map of epidermal growth factor receptor signaling. Molecular Systems Biology 1 (May 2005)
36. Weinberg, R.A.: The Biology of Cancer. Garland Science (June 2006)
37. Hynes, N., Lane, H.: ERBB receptors and cancer: the complexity of targeted inhibitors. Nature Reviews Cancer 5(5), 341–354 (2005)
38. Milner, R.: Bigraphical reactive systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 16–35. Springer, Heidelberg (2001)
39. Winskel, G.: An introduction to event structures. In: REX Workshop 1988. LNCS, Springer, Heidelberg (1989)

5 Appendix

5.1 κ, Briefly

In section 2.1 we introduced κ by means of an example. Here we provide some formal definitions to fix concepts more precisely and convey a sense of what we have implemented.

Agents. Let $\mathcal{A}$ be a countable set of names, $\mathcal{S}$ a countable set of sites, and $\mathbb{V}$ a finite set of values. An *agent* is a tuple consisting of an agent name $\lambda(a) \in \mathcal{A}$, a finite set of sites $\sigma(a) \subseteq \mathcal{S}$, and a partial valuation $\mu(a)$ in $\mathbb{V}^{\sigma(a)}$ assigning values to some of the agent's sites, and called the agent's internal state. In the context of signalling, agents are typically proteins (but they need not be).

Solution of agents. By a *solution* we usually have container of molecules in mind. In the case of chemistry proper, agents would be atoms and molecules are atoms connected in particular ways. Likewise, in biological signalling, agents are usually proteins and connected proteins (proteins that are noncovalently bound to one another) are complexes. More precisely, a solution S is a set of agents, together with a partial matching on the set $\sum_{a \in S} \sigma(a)$ of all agent sites. The matching specifies how agents are connected through their sites, but no site can be connected twice. One writes $(a, i), (b, j) \in S$ to express the fact that sites i, j in agents a, b are connected in S.

A solution is essentially a graph whose nodes are agents and whose edges are bonds between agents. Consequently, graph-theoretic notions such as subgraph, connected component, path, etc. apply. We can think of a connected component as a *complex* of proteins – the nodes of the component – bound to one another as specified by the edges of the component.

A *signature* map $\Sigma : \mathcal{A} \to \wp(\mathcal{S})$ is an assignment of a finite set of sites to each agent name. We assume such a signature map to be fixed once and for all, and consider only solutions S such that for all $a \in S$, $\sigma(a) \subseteq \Sigma(\lambda(a))$. An agent a is said to be complete if $\sigma(a) = \Sigma(\lambda(a))$, and likewise a solution S is said to be complete if all its agents are.

Rules. The examples of Subsection 2.1 introduced a rule as a transformation of the graph of agents specified on the left-hand-side (lhs) of the rule into the graph specified on its right-hand-side (rhs). Since the graph on the lhs can occur in many ways within a solution, we refer to it as a “pattern”. A solution is a graph, and so the lhs of a rule can be viewed as a solution as well (usually a small one compared to the solution that represents the whole system). A rule then is as a solution together with an action transforming it. The application of a rule to a system means first identifying an embedding of the rule's solution (the lhs pattern) in the solution representing the system and then applying the action to that location. This is made precise in the following by first defining the notion of embedding.

A map ϕ between solutions S and T is an *embedding* if it is an injection on agents, that preserves names, sites, internal states, and preserves and reflects edges; that is to say for all a, $b \in S$, i, $j \in \mathcal{S}$:

$$\begin{aligned}
&\phi(a) = \phi(b) \Rightarrow a = b \\
&\lambda_S(a) = \lambda_T(\phi(a)) \\
&\sigma_S(a) \subseteq \sigma_T(\phi(a)) \\
&\mu_S(a)(i) = v \Rightarrow \mu_T(\phi(a))(i) = v \\
&(a, i), (b, j) \in S \Leftrightarrow (\phi(a), i), (\phi(b), j) \in T
\end{aligned}$$

Hereafter, whenever we write $\phi : S \to T$ we mean to say that ϕ is an embedding, we also write $cod(\phi)$ for the set of sites in T which are in the image of ϕ, and $\mathcal{J}(S, T)$ for the set of all embeddings of S into T.

A *rule* is a pair (S, α), where S is a solution (the left-hand-side in the notation of Subsection 2.1) and an action α over S (the rule action). An *atomic action* changes the value of some site, or creates/deletes an edge between two sites, or creates/deletes an agent. An *action* on S is a sequence of atomic actions. A rule (S, α) is said to be *atomic*, if α is atomic. It is said to have *arity* n, if S has n connected components, in which case any $\phi : S \to T$ decomposes into n embeddings, one per connected component of S, which we call ϕ's *components*.

The number of all embeddings of the rule's S into the system T, $|\mathcal{J}(S,T)|$, plays an important role in the probabilistic simulation of a system specified by a concrete set of agents and complexes (the solution T) and a set of rules, as sketched in Subsection 2.3.

Causation (activation) and conflict (inhibition) between events (rules). Given an embedding $\phi : S \to T$, we write $\phi(\alpha) \cdot T$ for the result of α on T via ϕ. We say that a site in T is *modified* by $\phi(\alpha)$ if its internal state, or its connections are. A rule set R defines a labelled transition relation over complete solutions:

$$T \longrightarrow_{\phi}^{r} \phi(\alpha(r)) \cdot T$$

with $r = (S(r), \alpha(r)) \in R$, and ϕ an embedding from $S(r)$ into T. An *event* (r, ϕ) consists in identifying an embedding ϕ of the rule pattern $S(r)$ into a complete solution T, and applying the rule action $\alpha(r)$ along ϕ.

Let $\mathcal{E}(T)$ denote the set of events in T. We can define the notions of conflict and causation between events by comparing T with $\phi(\alpha(r)) \cdot T$. Given an event $(r, \phi) \in \mathcal{E}(T)$, we say that (r, ϕ) *conflicts* with (s, ψ), if $(s, \psi) \in \mathcal{E}(T) \setminus \mathcal{E}(\phi(\alpha(r)) \cdot T)$. We say that (r, ϕ) *causes* (s, ψ), if $(s, \psi) \in \mathcal{E}(\phi(\alpha(r)) \cdot T) \setminus \mathcal{E}(T)$.

Unlike in the classical notion of event structure [39], conflict here is not symmetric. It is quite possible that ψ does not conflict with ϕ, while ϕ conflicts with ψ.

The following definition is useful in projecting the definitions of conflict and causation at the level of events to the level of rules, where we refer to them as inhibition and activation, respectively. Given an event $e = (r, \phi)$ in $\mathcal{E}(T)$, the *negative support* of e, written $\lfloor e \rfloor_-$, is the set of sites in T which $\phi(\alpha(r))$ erases or modifies. Similarly, the *positive support* of e, written $\lfloor e \rfloor_+$, is the set of sites in $\phi(\alpha(r)) \cdot T$ which $\phi(\alpha(r))$ creates or modifies.

Using the notion of support, we can formulate necessary conditions for conflict and causation between events. Consider an event $e = (r, \phi)$ in $\mathcal{E}(T)$. If e conflicts with $(s, \psi) \in \mathcal{E}(T)$, then $\mathit{cod}(\psi) \cap \lfloor e \rfloor_- \neq \emptyset$. If e causes $(s, \psi) \in \mathcal{E}(\phi(\alpha(r)) \cdot T)$, then $\mathit{cod}(\psi) \cap \lfloor e \rfloor_+ \neq \emptyset$. At the level of rules, we say that r *inhibits* s if for some T, ϕ, ψ, $\mathit{cod}(\psi) \cap \lfloor e \rfloor_- \neq \emptyset$, and one says r *activates* s if for some T, ϕ, ψ, $\mathit{cod}(\psi) \cap \lfloor e \rfloor_+ \neq \emptyset$.

The notions of inhibition and activation between rules should not be confused with the notions of conflict and causation between events from which they are derived. The relations of inhibition and activation are static relationships between rules and can be computed once and for all for a given set of rules.

5.2 The Rule Set of the EGFR Model

This appendix contains the κ-representation of the Schoeberl et al. (2002) EGF receptor model [12], which introduced receptor internalisation, and combines it with the negative feedback mechanism described in the earlier Brightman & Fell (2000) model [11]. A useful review of these and many other models is provided in Ref. [13]. Refactoring those models in κ involved mining the literature and various databases to obtain the missing domain related information.

Rule names used in the main text and the figures of the paper are defined below. Certain rules use a shorthand '`!_`' notation, to mean that a site is 'bound to something' but the rule does not test anything further than that. This is a convenient way to shorten rules.

Rate constants of rules were set to 1 by default, except for internalisation rules. The numerical experiments summarized in Fig.9 and Fig.10 varied pertinent rate constants as explained above. The rules are presented roughly in the order implied by the ERK activation story shown in Fig. 8. The initial state used in our EGFR simulations is also shown below.

Activating receptor dimers

```
# external dimers:
'EGF_EGFR'  EGF(r~ext), EGFR(L~ext,CR) <-> EGF(r~ext!1), EGFR(L~ext!1,CR)
'EGFR_EGFR' EGFR(L~ext!_,CR), EGFR(L~ext!_,CR) <->
            EGFR(L~ext!_,CR!1), EGFR(L~ext!_,CR!1)
# simplified phosphorylation (internal or external)
'EGFR@992'  EGFR(CR!_,Y992~u) -> EGFR(CR!_,Y992~p)
'EGFR@1068' EGFR(CR!_,Y1068~u) -> EGFR(CR!_,Y1068~p)
'EGFR@1148' EGFR(CR!_,Y1148~u) -> EGFR(CR!_,Y1148~p)
# simplified dephosphorylation (internal or external)
'992_op'    EGFR(Y992~p) -> EGFR(Y992~u)
'1068_op'   EGFR(Y1068~p) -> EGFR(Y1068~u)
'1148_op'   EGFR(Y1148~p) -> EGFR(Y1148~u)
```

Internalization, degradation and recycling

```
# internalization:
'int_monomer'  EGF(r~ext!1), EGFR(L~ext!1,CR) ->
               EGF(r~int!1), EGFR(L~int!1,CR) @ 0.02
'int_dimer'    EGF(r~ext!1), EGFR(L~ext!1,CR!2),
               EGF(r~ext!3), EGFR(L~ext!3,CR!2) ->
               EGF(r~int!1), EGFR(L~int!1,CR!2),
               EGF(r~int!3), EGFR(L~int!3,CR!2) @ 0.02
# dissociation:
'EGFR_EGFR_op' EGFR(L~int!_,CR!1), EGFR(L~int!_,CR!1) ->
               EGFR(L~int!_,CR), EGFR(L~int!_,CR)
'EGF_EGFR_op'  EGF(r~int!1), EGFR(L~int!1,CR) ->
               EGF(r~int), EGFR(L~int,CR)
# degradation:
'deg_EGF'      EGF(r~int) ->
```

```
'deg_EGFR'      EGFR(L~int,CR) ->
# recycling:
'rec_EGFR'      EGFR(L~int,Y992~u,Y1068~u,Y1148~u) ->
                EGFR(L~ext,Y992~u,Y1068~u,Y1148~u)
```

SoS and RasGAP recruitment

```
'EGFR_RasGAP'  EGFR(Y992~p), RasGAP(SH2) <-> EGFR(Y992~p!1), RasGAP(SH2!1)
'EGFR_Grb2'     EGFR(Y1068~p), Grb2(SH2) <-> EGFR(Y1068~p!1), Grb2(SH2!1)
'Grb2_SoS'      Grb2(SH3), SoS(a,SS~u) ->
                Grb2(SH3!1), SoS(a!1,SS~u)
'Grb2_SoS_op'   Grb2(SH3!1), SoS(a!1) -> Grb2(SH3), SoS(a)
'EGFR_Shc'      EGFR(Y1148~p), Shc(PTB) <-> EGFR(Y1148~p!1), Shc(PTB!1)
'Shc_Grb2'      Shc(Y318~p), Grb2(SH2) <-> Shc(Y318~p!1), Grb2(SH2!1)
'Shc@318'       EGFR(CR!_,Y1148~p!1), Shc(PTB!1,Y318~u) ->
                EGFR(CR!_,Y1148~p!1), Shc(PTB!1,Y318~p)
'Shc@318_op'    Shc(Y318~p) -> Shc(Y318~u)
```

Activating Ras

```
# activate:
'long arm SoS_Ras'  EGFR(Y1148~p!1), Shc(PTB!1,Y318~p!2),
                    Grb2(SH2!2,SH3!3), SoS(a!3,b), Ras(S1S2~gdp) ->
                    EGFR(Y1148~p!1), Shc(PTB!1,Y318~p!2),
                    Grb2(SH2!2,SH3!3), SoS(a!3,b!4), Ras(S1S2~gdp!4)
'short arm SoS_Ras' EGFR(Y1068~p!1), Grb2(SH2!1,SH3!2),
                    SoS(a!2,b), Ras(S1S2~gdp) ->
                    EGFR(Y1068~p!1), Grb2(SH2!1,SH3!2),
                    SoS(a!2,b!3), Ras(S1S2~gdp!3)
'Ras GTP'           SoS(b!1), Ras(S1S2~gdp!1) -> SoS(b!1), Ras(S1S2~gtp!1)
'SoS_Ras_op'        SoS(b!1), Ras(S1S2!1) -> SoS(b), Ras(S1S2)

# deactivate:
'direct RasGAP_Ras' EGFR(Y992~p!1), RasGAP(SH2!1,s), Ras(S1S2~gtp) ->
                    EGFR(Y992~p!1), RasGAP(SH2!1,s!2), Ras(S1S2~gtp!2)
'Ras GDP'           RasGAP(s!1), Ras(S1S2~gtp!1) ->
                    RasGAP(s!1), Ras(S1S2~gdp!1)
'RasGAP_Ras_op'     RasGAP(s!1), Ras(S1S2!1) -> RasGAP(s), Ras(S1S2)
'intrinsic Ras GDP' Ras(S1S2~gtp) -> Ras(S1S2~gdp)
```

Activating Raf

```
# activation:
'Ras_Raf'       Ras(S1S2~gtp), Raf(x~u) -> Ras(S1S2~gtp!1), Raf(x~u!1)
'Raf'           Ras(S1S2~gtp!1), Raf(x~u!1) -> Ras(S1S2~gtp!1), Raf(x~p!1)
'Ras_Raf_op'    Ras(S1S2~gtp!1), Raf(x!1) -> Ras(S1S2~gtp), Raf(x)
# deactivation:
'PP2A1_Raf'     PP2A1(s), Raf(x~p) -> PP2A1(s!1), Raf(x~p!1)
'Raf_op'        PP2A1(s!1), Raf(x~p!1) -> PP2A1(s!1), Raf(x~u!1)
'PP2A1_Raf_op'  PP2A1(s!1), Raf(x!1) -> PP2A1(s), Raf(x)
```

Activating MEK

```
# activation:
'Raf_MEK@222'      Raf(x~p), MEK(S222~u) -> Raf(x~p!1), MEK(S222~u!1)
'MEK@222'          Raf(x~p!1), MEK(S222~u!1) -> Raf(x~p!1), MEK(S222~p!1)
'Raf_MEK@222_op'   Raf(x~p!1), MEK(S222!1) -> Raf(x~p), MEK(S222)
'Raf_MEK@218'      Raf(x~p), MEK(S218~u) -> Raf(x~p!1), MEK(S218~u!1)
'MEK@218'          Raf(x~p!1), MEK(S218~u!1) -> Raf(x~p!1), MEK(S218~p!1)
'Raf_MEK@218_op'   Raf(x~p!1), MEK(S218!1) -> Raf(x~p), MEK(S218)
# deactivation:
'PP2A2_MEK@222'    PP2A2(s), MEK(S222~p) -> PP2A2(s!1), MEK(S222~p!1)
'MEK@222_op'       PP2A2(s!1), MEK(S222~p!1) -> PP2A2(s!1), MEK(S222~u!1)
'PP2A2_MEK@222_op' PP2A2(s!1), MEK(S222!1) -> PP2A2(s), MEK(S222)
'PP2A2_MEK@218'    PP2A2(s), MEK(S218~p) -> PP2A2(s!1), MEK(S218~p!1)
'MEK@218_op'       PP2A2(s!1), MEK(S218~p!1) -> PP2A2(s!1), MEK(S218~u!1)
'PP2A2_MEK@218_op' PP2A2(s!1), MEK(S218!1) -> PP2A2(s), MEK(S218)
```

Activating ERK

```
# activation:
'MEK_ERK@185'    MEK(s,S218~p,S222~p), ERK(T185~u) ->
                 MEK(s!1,S218~p,S222~p), ERK(T185~u!1)
'ERK@185'        MEK(s!1,S218~p,S222~p), ERK(T185~u!1) ->
                 MEK(s!1,S218~p,S222~p), ERK(T185~p!1)
'MEK_ERK@185_op' MEK(s!1), ERK(T185!1) -> MEK(s), ERK(T185)
'MEK_ERK@187'    MEK(s,S218~p,S222~p), ERK(Y187~u) ->
                 MEK(s!1,S218~p,S222~p), ERK(Y187~u!1)
'ERK@187'        MEK(s!1,S218~p,S222~p), ERK(Y187~u!1) ->
                 MEK(s!1,S218~p,S222~p), ERK(Y187~p!1)
'MEK_ERK@187_op' MEK(s!1), ERK(Y187!1) -> MEK(s), ERK(Y187)
# deactivation:
'MKP_ERK@185'    MKP3(s), ERK(T185~p) -> MKP3(s!1), ERK(T185~p!1)
'ERK@185_op'     MKP3(s!1), ERK(T185~p!1) -> MKP3(s!1), ERK(T185~u!1)
'MKP_ERK@185_op' MKP3(s!1), ERK(T185!1) -> MKP3(s), ERK(T185)
'MKP_ERK@187'    MKP3(s), ERK(Y187~p) -> MKP3(s!1), ERK(Y187~p!1)
'ERK@187_op'     MKP3(s!1), ERK(Y187~p!1) -> MKP3(s!1), ERK(Y187~u!1)
'MKP_ERK@187_op' MKP3(s!1), ERK(Y187!1) -> MKP3(s), ERK(Y187)
```

Deactivating SoS

```
'SoS_ERK'    SoS(SS~u), ERK(s,T185~p,Y187~p) ->
             SoS(SS~u!1), ERK(s!1,T185~p,Y187~p)
'SoS_ERK_op' SoS(SS!1), ERK(s!1) -> SoS(SS), ERK(s)
# feedback creation
'SoS@SS'     SoS(SS~u!1), ERK(s!1,T185~p,Y187~p) ->
             SoS(SS~p!1), ERK(s!1,T185~p,Y187~p)
# feedback recovery
'SoS@SS_op'  SoS(SS~p) -> SoS(SS~u)

%init: 10*(EGF(r~ext))
+ 100*(EGFR(L~ext,CR,Y992~u,Y1068~u,Y1148~u))
+ 100*(Shc(PTB,Y318~u))
+ 100*(Grb2(SH2,SH3!1),SoS(a!1,b,SS~u))
+ 200*(RasGAP(SH2,s))
+ 100*(Ras(S1S2~gdp))
+ 100*(Raf(x~u))
+ 25*(PP2A1(s))
+ 50*(PP2A2(s))
+ 200*(MEK(s,S222~u,S218~u))
+ 200*(ERK(s,T185~u,Y187~u))
+ 50*(MKP3(s))
```

Making Random Choices Invisible to the Scheduler*

Konstantinos Chatzikokolakis and Catuscia Palamidessi

INRIA and LIX, École Polytechnique, France
{kostas,catuscia}@lix.polytechnique.fr

Abstract. When dealing with process calculi and automata which express both nondeterministic and probabilistic behavior, it is customary to introduce the notion of scheduler to resolve the nondeterminism. It has been observed that for certain applications, notably those in security, the scheduler needs to be restricted so not to reveal the outcome of the protocol's random choices, or otherwise the model of adversary would be too strong even for "obviously correct" protocols. We propose a process-algebraic framework in which the control on the scheduler can be specified in syntactic terms, and we show how to apply it to solve the problem mentioned above. We also consider the definition of (probabilistic) may and must preorders, and we show that they are precongruences with respect to the restricted schedulers. Furthermore, we show that all the operators of the language, except replication, distribute over probabilistic summation, which is a useful property for verification.

1 Introduction

Security protocols, in particular those for anonymity and fair exchange, often use randomization to achieve their targets. Since they usually involve more than one agent, they also give rise to concurrent and interactive activities that can be best modeled by nondeterminism. Thus it is convenient to specify them using a formalism which is able to represent both *probabilistic* and *nondeterministic* behavior. Formalisms of this kind have been explored in both Automata Theory [1,2,3,4,5] and in Process Algebra [6,7,8,9,10,11]. See also [12,13] for comparative and more inclusive overviews.

Due to the presence of nondeterminism, in such formalisms it is not possible to define the probability of events in *absolute* terms. We need first to decide how each nondeterministic choice during the execution will be resolved. This decision function is called *scheduler*. Once the scheduler is fixed, the behavior of the system (*relatively* to the given scheduler) becomes fully probabilistic and a probability measure can be defined following standard techniques.

It has been observed by several researchers that in security the notion of scheduler needs to be restricted, or otherwise any secret choice of the protocol

* This work has been partially supported by the INRIA DREI Équipe Associée PRINTEMPS and by the INRIA ARC project ProNoBiS.

could be revealed by making the choice of the scheduler depend on it. This issue was for instance one of the main topics of discussion at the panel of CSFW 2006. We illustrate it here with an example on anonymity. We use the standard CCS notation, plus a construct of probabilistic choice $P +_p Q$ representing a process that evolves into P with probability p and into Q with probability $1 - p$.

The system *Sys* consists of a receiver R and two senders S, T communicating via private channels a, b respectively. Which of the two senders is successful is decided probabilistically by R. After reception, R sends a signal *ok*.

$$R \triangleq a.\overline{ok}.0 \ +_{0.5} \ b.\overline{ok}.0 \qquad S \triangleq \bar{a}.0 \qquad T \triangleq \bar{b}.0 \qquad Sys \triangleq (\nu a)(\nu b)(R \mid S \mid T)$$

The signal *ok* is not private, but since it is the same in both cases, in principle an external observer should not be able to infer from it the identity of the sender (S or T). So the system should be anonymous. However, consider a team of two attackers A and B defined as

$$A \triangleq \ ok.\bar{s}.0 \qquad B \triangleq \ ok.\bar{t}.0$$

and consider the parallel composition $Sys \mid A \mid B$. We have that, under certain schedulers, the system is no longer anonymous. More precisely, a scheduler could leak the identity of the sender via the channels s, t by forcing R to synchronize with A on *ok* if R has chosen the first alternative, and with B otherwise. This is because in general a scheduler can see the whole history of the computation, in particular the random choices, even those which are supposed to be private.

There is another issue related to verification: a private choice has certain algebraic properties that would be useful in proving equivalences between processes. In fact, if the outcome of a choice remains private, then it should not matter at which point of the execution the process makes such choice, until it actually uses it. Consider for instance A and B defined as follows

$$\begin{aligned} A \triangleq a(x).([x = 0]&\overline{ok} \\ &+_{0.5} \\ [x = 1]&\overline{ok}) \end{aligned} \qquad\qquad \begin{aligned} B \triangleq a(x).[x = 0]&\overline{ok} \\ &+_{0.5} \\ a(x).[x = 1]&\overline{ok} \end{aligned}$$

Process A receives a value and then decides randomly whether it will accept the value 0 or 1. Process B does exactly the same thing except that the choice is performed before the reception of the value. If the random choices in A and B are private, intuitively we should have that A and B are equivalent ($A \approx B$). This is because it should not matter whether the choice is done before or after receiving a message, as long as the outcome of the choice is completely invisible to any other process or observer. However, consider the parallel context $C = \overline{a}0 \mid \overline{a}1$. Under any scheduler A has probability at most $1/2$ to perform $\overline{ok}$. With B, on the other hand, the scheduler can choose between $\overline{a}0$ and $\overline{a}1$ based on the outcome of the probabilistic choice, thus making the maximum probability of $\overline{ok}$ equal to 1. The execution trees of $A \mid C$ and $B \mid C$ are shown in Figure 1.

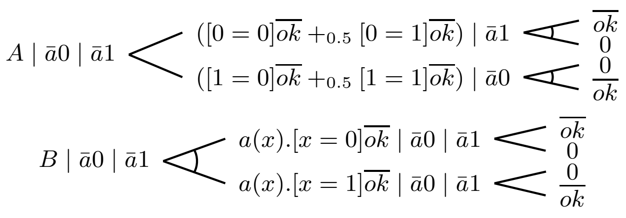

$A \mid \bar{a}0 \mid \bar{a}1$
$([0=0]\overline{ok} +_{0.5} [0=1]\overline{ok}) \mid \bar{a}1$
$([1=0]\overline{ok} +_{0.5} [1=1]\overline{ok}) \mid \bar{a}0$
$\overline{ok}$
0
0
$\overline{ok}$
$B \mid \bar{a}0 \mid \bar{a}1$
$a(x).[x=0]\overline{ok} \mid \bar{a}0 \mid \bar{a}1$
$a(x).[x=1]\overline{ok} \mid \bar{a}0 \mid \bar{a}1$
$\overline{ok}$
0
0
$\overline{ok}$

Fig. 1. Execution trees for $A \mid C$ and $B \mid C$

In general when $+_p$ represents a private choice we would like to have

$$C[P +_p Q] \approx C[\tau.P] +_p C[\tau.Q] \tag{1}$$

for all processes P, Q and all contexts C *not containing replication (or recursion)*. In the case of replication the above cannot hold since $!(P +_p Q)$ makes available each time the choice between P and Q, while $(!\tau.P) +_p (!\tau.Q)$ chooses once and for all which of the two (P or Q) should be replicated. Similarly for recursion. The reason why we need a τ is explained in Section 5.

The algebraic property (1) expresses in an abstract way the privacy of the probabilistic choice. Moreover, this property is also useful for the verification of security properties. The interested reader can find in [14] an example of application to a fair exchange protocol.

We propose a process-algebraic approach to the problem of hiding the outcome of random choices. Our framework is based on CCS$_p$, a calculus obtained by adding to CCS an internal probabilistic choice. This calculus is a variant of the one studied in [11], the main differences being that we use replication instead of recursion, and we lift some restrictions that were imposed in [11] to obtain a complete axiomatization. The semantics of CCS$_p$ is given in terms of *simple probabilistic automata* [4,7].

In order to limit the power of the scheduler, we extend CCS$_p$ with terms representing the scheduler explicitly. The latter interacts with the original processes via a labeling system. This will allow to specify which choices should be visible to schedulers, and which ones should not. We call the extended calculus CCS$_\sigma$.

We then adapt the standard notions of probabilistic testing preorders to CCS$_\sigma$, and we show that they are precongruences with respect to all the operators except the $+$. We also prove that, under suitable conditions on the labelings of C, $\tau.P$ and $\tau.Q$, CCS$_\sigma$ satisfies the property expressed by (1), where $\approx$ is probabilistic testing equivalence.

We apply our approach to an anonymity example (the Dining Cryptographers Protocol, DCP). We also briefly outline how to extend CCS$_\sigma$ so to allow the definition of private nondeterministic choice, and we apply it to the DCP with nondeterministic master. To our knowledge this is the first formal treatment of the scheduling problem in DCP and the first formalization of a nondeterministic master for the (probabilistic) DCP.

See `www.lix.polytechnique.fr/~catuscia/papers/Scheduler/report.pdf` for the report version of this paper, containing more details.

1.1 Related Work

The works that are most closely related to ours are [15,16,17]. In [15,16] the authors consider probabilistic automata and introduce a restriction on the scheduler to the purpose of making them suitable to applications in security protocols. Their approach is based on dividing the actions of each component of the system in equivalence classes (*tasks*). The order of execution of different tasks is decided in advance by a so-called *task scheduler*. The remaining nondeterminism within a task is resolved by a second scheduler, which models the standard *adversarial scheduler* of the cryptographic community. This second entity has limited knowledge about the other components: it sees only the information that they communicate during execution.

In [17] the authors define a notion of admissible scheduler by introducing an equivalence relation on the nodes of the execution tree, and requiring that an admissible scheduler maps two equivalent nodes into bisimilar steps. Both our paper and [17] have developed, independently, the solution to the problem of the scheduler in the Dining Cryptographers as an example of application to security.

Our approach is in a sense *dual* to the above ones. Instead of defining a restriction on the class of schedulers, we propose a way of controlling the scheduler at the syntactic level. More precisely, we introduce labels in process terms, and we use them to represent both the nodes of the execution tree and the next action or step to be scheduled. We make two nodes indistinguishable to schedulers, and hence the choice between them private, by associating to them the same label. Furthermore, in contrast with [15,16], our "equivalence classes" (schedulable actions with the same label) can change dynamically, because the same action can be associated to different labels during the execution. However we don't know at the moment whether this difference determines a separation in the expressive power.

Another work along these lines is [18], which uses partitions on the state-space to obtain partial-information schedulers. However in that paper the authors consider a synchronous parallel composition, so the setting is rather different.

2 Preliminaries

We recall here some notions about the simple probabilistic automata and CCS$_p$.

2.1 Simple Probabilistic Automata [4,7]

A *discrete probability measure* over a set X is a function $\mu : 2^X \mapsto [0,1]$ such that $\mu(X) = 1$ and $\mu(\cup_i X_i) = \sum_i \mu(X_i)$ where X_i is a countable family of pairwise disjoint subsets of X. We denote the set of all discrete probability measures over X by $Disc(X)$. For $x \in X$, we denote by $\delta(x)$ (the *Dirac measure* on x) the probability measure that assigns probability 1 to $\{x\}$. We also denote by $\sum_i [p_i]\mu_i$ the probability measure obtained as a convex sum of the measures μ_i.

A (*simple*) *probabilistic automaton* is a tuple $(S, q, A, \mathcal{D})$ where S is a set of states, $q \in S$ is the *initial state*, A is a set of actions and $\mathcal{D} \subseteq S \times A \times Disc(S)$ is a *transition relation*. Intuitively, if $(s, a, \mu) \in \mathcal{D}$ then there is a transition (*step*) from the state s performing the action a and leading to a distribution μ over the states of the automaton. The idea is that the choice of the transition in $\mathcal{D}$ is performed nondeterministically, and the choice of the target state among the ones allowed by μ (i.e. the q's such that $\mu(q) > 0$) is performed probabilistically.

A probabilistic automaton M is *fully probabilistic* if from each state of M there is at most one transition available. An execution α of a probabilistic automaton is a (possibly infinite) sequence $s_0 a_1 s_1 a_2 s_2 \ldots$ of alternating states and actions, such that $q = s_0$, and for each i $(s_i, a_{i+1}, \mu_i) \in \mathcal{D}$ and $\mu_i(s_{i+1}) > 0$ hold. We will use $lstate(\alpha)$ to denote the last state of a finite execution α, and $exec^*(M)$ and $exec(M)$ to represent the set of all the finite and of all the executions of M, respectively.

A *scheduler* of a probabilistic automaton $M = (S, q, A, \mathcal{D})$ is a function

$$\zeta : exec^*(M) \mapsto \mathcal{D}$$

such that $\zeta(\alpha) = (s, a, \mu) \in \mathcal{D}$ implies that $s = lstate(\alpha)$. The idea is that a scheduler selects a transition among the ones available in $\mathcal{D}$, basing its decision on the history of the execution. The *execution tree* of M relative to the scheduler ζ, denoted by $etree(M, \zeta)$, is a fully probabilistic automaton $M' = (S', q', A', \mathcal{D}')$ such that $S' \subseteq exec(M)$, $q' = q$, $A' = A$, and $(\alpha, a, \mu') \in \mathcal{D}'$ if and only if $\zeta(\alpha) = (lstate(\alpha), a, \mu)$ for some μ and $\mu'(\alpha a s) = \mu(s)$. Intuitively, $etree(M, \zeta)$ is produced by unfolding the executions of M and resolving all deterministic choices using ζ. Note that $etree(M, \zeta)$ is a simple and fully probabilistic automaton.

2.2 CCS with Internal Probabilistic Choice

Let a range over a countable set of *channel names*. The syntax of CCS$_p$ is:

$$\begin{array}{rcll} \alpha & ::= & a \mid \bar{a} \mid \tau & \textbf{prefixes} \\ P, Q & ::= & \alpha.P \mid P \mid Q \mid P + Q \mid \sum_i p_i P_i \mid (\nu a)P \mid\ !P \mid 0 & \textbf{processes} \end{array}$$

The term $\sum_i p_i P_i$ represents an *internal probabilistic choice*, all the rest is standard. We will also use the notation $P_1 +_p P_2$ to represent a binary sum $\sum_i p_i P_i$ with $p_1 = p$ and $p_2 = 1 - p$.

The semantics of a CCS$_p$ term is a probabilistic automaton defined according to the rules in Figure 2. We write $s \xrightarrow{a} \mu$ when (s, a, μ) is a transition of the probabilistic automaton. We also denote by $\mu \mid Q$ the measure μ' such that $\mu'(P \mid Q) = \mu(P)$ for all processes P and $\mu'(R) = 0$ if R is not of the form $P \mid Q$. Similarly $(\nu a)\mu = \mu'$ such that $\mu'((\nu a)P) = \mu(P)$. A transition of the form $P \xrightarrow{a} \delta(P')$, i.e. a transition having for target a Dirac measure, corresponds to a transition of a non-probabilistic automaton.

3 A Variant of CCS with Explicit Scheduler

In this section we present CCS$_\sigma$, a variant of CCS in which the scheduler is explicit, in the sense that it has a specific syntax and its behavior is defined

$$\text{ACT}\ \frac{}{\alpha.P \xrightarrow{\alpha} \delta(P)} \qquad \text{RES}\ \frac{P \xrightarrow{\alpha} \mu \quad \alpha \neq a, \overline{a}}{(\nu a)P \xrightarrow{\alpha} (\nu a)\mu}$$

$$\text{SUM1}\ \frac{P \xrightarrow{\alpha} \mu}{P + Q \xrightarrow{\alpha} \mu} \qquad \text{PAR1}\ \frac{P \xrightarrow{\alpha} \mu}{P \mid Q \xrightarrow{\alpha} \mu \mid Q}$$

$$\text{COM}\ \frac{P \xrightarrow{a} \delta(P') \quad Q \xrightarrow{\overline{a}} \delta(Q')}{P \mid Q \xrightarrow{\tau} \delta(P' \mid Q')} \qquad \text{PROB}\ \frac{}{\sum_i p_i P_i \xrightarrow{\tau} \sum_i [p_i]\delta(P_i)}$$

$$\text{REP1}\ \frac{P \xrightarrow{\alpha} \mu}{!P \xrightarrow{\alpha} \mu \mid !P} \qquad \text{REP2}\ \frac{P \xrightarrow{a} \delta(P_1) \quad P \xrightarrow{\overline{a}} \delta(P_2)}{!P \xrightarrow{\tau} \delta(P_1 \mid P_2 \mid !P)}$$

Fig. 2. The semantics of CCS$_p$. SUM1 and PAR1 have corresponding right rules SUM2 and PAR2, omitted for simplicity.

$I ::= 0I \mid 1I \mid \epsilon$	**label indexes**	$S, T ::=$	**scheduler**
$L ::= l^I$	**labels**	$L.S$	schedule single action
		$\mid (L, L).S$	synchronization
$P, Q ::=$	**processes**	$\mid$ **if** L **then** S **else** S	label test
$L{:}\alpha.P$	prefix		
$\mid P \mid Q$	parallel		
$\mid P + Q$	nondeterm. choice	$\mid 0$	nil
$\mid L{:}\sum_i p_i P_i$	internal prob. choice		
$\mid (\nu a)P$	restriction	$CP ::= P \parallel S$	**complete process**
$\mid\ !P$	replication		
$\mid L{:}0$	nil		

Fig. 3. The syntax of the core CCS$_\sigma$

by the operational semantics of the calculus. Processes in CCS$_\sigma$ contain labels that allow us to refer to a particular sub-process. A scheduler also behaves like a process, using however a different syntax, and its purpose is to guide the execution of the main process using the labels that the latter provides. A *complete process* is a process running in parallel with a scheduler.

3.1 Syntax

Let a range over a countable set of *channel names* and l over a countable set of *atomic labels*. The syntax of CCS$_\sigma$, shown in Figure 3, is the same as the one of CCS$_p$ except for the presence of labels. These are used to select the subprocess which "performs" a transition. Since only the operators with an initial rule can originate a transition, we only need to assign labels to the prefix and to the probabilistic sum. For reasons explained later, we also put labels on 0, even though this is not required for scheduling transitions. We use labels of the form l^s where l is an atomic label and the index s is a finite string of 0 and 1, possibly

$$\text{ACT}\ \frac{}{l{:}\alpha.P \parallel l.S \xrightarrow{\alpha} \delta(P \parallel S)} \qquad \text{RES}\ \frac{P \parallel S \xrightarrow{\alpha} \mu \quad \alpha \neq a, \overline{a}}{(\nu a)P \parallel S \xrightarrow{\alpha} (\nu a)\mu}$$

$$\text{SUM1}\ \frac{P \parallel S \xrightarrow{\alpha} \mu}{P + Q \parallel S \xrightarrow{\alpha} \mu} \qquad \text{PAR1}\ \frac{P \parallel S \xrightarrow{\alpha} \mu}{P \mid Q \parallel S \xrightarrow{\alpha} \mu \mid Q}$$

$$\text{COM}\ \frac{P \parallel l_1 \xrightarrow{a} \delta(P' \parallel 0) \qquad Q \parallel l_2 \xrightarrow{\overline{a}} \delta(Q' \parallel 0)}{P \mid Q \parallel (l_1, l_2).S \xrightarrow{\tau} \delta(P' \mid Q' \parallel S)}$$

$$\text{REP1}\ \frac{P \parallel S \xrightarrow{\alpha} \mu}{!P \parallel S \xrightarrow{\alpha} \rho_0(\mu) \mid \rho_1(!P)} \qquad \text{PROB}\ \frac{}{l{:}\sum_i p_i P_i \parallel l.S \xrightarrow{\tau} \sum_i [p_i]\delta(P_i \parallel S)}$$

$$\text{REP2}\ \frac{P \parallel l_1 \xrightarrow{a} \delta(P_1 \parallel 0) \qquad P \parallel l_2 \xrightarrow{\overline{a}} \delta(P_2 \parallel 0)}{!P \parallel (l_1, l_2).S \xrightarrow{\tau} \delta(\rho_0(P_1) \mid \rho_{10}(P_2) \mid \rho_{11}(!P) \parallel S)}$$

$$\text{IF1}\ \frac{l \in tl(P) \qquad P \parallel S_1 \xrightarrow{\alpha} \mu}{P \parallel \textbf{if } l \textbf{ then } S_1 \textbf{ else } S_2 \xrightarrow{\alpha} \mu} \qquad \text{IF2}\ \frac{l \notin tl(P) \qquad P \parallel S_2 \xrightarrow{\alpha} \mu}{P \parallel \textbf{if } l \textbf{ then } S_1 \textbf{ else } S_2 \xrightarrow{\alpha} \mu}$$

Fig. 4. The semantics of CCS$_\sigma$. SUM1 and PAR1 have corresponding right rules SUM2 and PAR2, omitted for simplicity.

empty. With a slight abuse of notation we will sometimes use l to denote an arbitrary label, not necessarily atomic. Indexes are used to avoid multiple copies of the same label in case of replication, which occurs dynamically due to the bang operator.

A scheduler selects a sub-process for execution on the basis of its label, so we use $l.S$ to represent a scheduler that selects the process with label l and continues as S. In the case of synchronization we need to select two processes simultaneously, hence we need a scheduler of the form $(l_1, l_2).S$. Using **if-then-else** the scheduler can test whether a label is available in the process (in the top-level) and act accordingly. A complete process is a process put in parallel with a scheduler, for example $l_1 : a.l_2 : b \parallel l_1.l_2$. Note that for processes with an infinite execution path we need schedulers of infinite length.

3.2 Semantics

The operational semantics CCS$_\sigma$ is given in terms of probabilistic automata defined according to the rules shown in Figure 4.

ACT is the basic communication rule. In order for $l : \alpha.P$ to perform α, the scheduler should select this process for execution, so it needs to be of the form $l.S$. After the execution the complete process will continue as $P \parallel S$. The rules RES, SUM1, COM, PROB and PAR1 should be clear. Similarly to the Section 2.2, we denote by $(\nu a)\mu$ the measure μ' such that $\mu'((\nu a)P \parallel S) = \mu(P \parallel S)$ and $\mu \mid Q$ denotes the measure μ' such that $\mu'(P \mid Q \parallel S) = \mu(P \parallel S)$. Note that in

SUM1, PAR1 the scheduler resolves the non-deterministic choice by selecting a label inside P. In PROB, however, the scheduler cannot affect the outcome of the probabilistic choice, it can only schedule the choice itself.

REP1 and REP2 model replication. The rules are the same as in CCS$_p$, with the addition of a re-labeling operator ρ_k. The reason for this is that we want to avoid ending up with multiple copies of the same label as the result of replication, since this would create ambiguities in scheduling as explained in Section 3.3. $\rho_k(P)$ replaces all labels l^s inside P with l^{sk}, and it is defined as

$$\rho_k(l^s{:}\alpha.P) = l^{sk}{:}\alpha.\rho_k(P)$$
$$\rho_k(l^s{:}\textstyle\sum_i p_i P_i) = l^{sk}{:}\textstyle\sum_i p_i \rho_k(P_i)$$

and homomorphically on the other operators (for instance $\rho_k(P \mid Q) = \rho_k(P) \mid \rho_k(Q)$). We also denote by $\rho_k(\mu)$ the measure μ' such that $\mu'(\rho_k(P) \parallel S) = \mu(P \parallel S)$. Note that we relabel only the resulting process, not the continuation of the scheduler: there is no need for relabeling the scheduler since we are free to choose the continuation as we please.

Finally **if-then-else** allows the scheduler to adjust its behaviour based on the labels that are available in P. $tl(P)$ gives the set of top-level labels of P and is defined as $tl(l{:}\alpha.P) = tl(l{:}\sum_i p_i P_i) = tl(l{:}0) = \{l\}$ and as the union of the top-level labels of all sub-processes for the other operators. Then **if** l **then** S_1 **else** S_2 behaves like S_1 if l is available in P and as S_2 otherwise. This is needed when P is the outcome of a probabilistic choice, as discussed in Section 4.

3.3 Deterministic Labelings

The idea in CCS$_\sigma$ is that a *syntactic* scheduler will be able to completely resolve the nondeterminism of the process, without needing to rely on a *semantic* scheduler at the level of the automaton. This means that the execution of a process in parallel with a scheduler should be fully probabilistic. To achieve this we will impose a condition on the labels that we can use in CCS$_\sigma$ processes. A *labeling* is an assignment of labels to the prefixes, the probabilistic sums and the 0s of a process. We will require all labelings to be *deterministic* in the following sense.

Definition 1. *A labeling of a process P is* deterministic *iff for all schedulers S there is only one transition rule $P \parallel S \xrightarrow{\alpha} \mu$ that can be applied and the labelings of all processes P' such that $\mu(P' \parallel S') > 0$ are also deterministic.*

A labeling is *linear* iff all labels are pairwise distinct. We can show that linear labelings are preserved by transitions, which leads to the following proposition.

Proposition 1. *A linear labeling is deterministic.*

There are labelings that are deterministic without being linear. In fact, such labelings will be the means by which we hide information from the scheduler. However, the property of being deterministic is crucial since it implies that the scheduler will resolve all the nondeterminism of the process.

Proposition 2. *Let P be a CCS$_\sigma$ process with a deterministic labeling. Then for all schedulers S, the automaton produced by $P \parallel S$ is fully probabilistic.*

4 Expressiveness of the Syntactic Scheduler

CCS$_\sigma$ with deterministic labelings allows us to separate probabilities from nondeterminism: a process in parallel with a scheduler behaves in a fully probabilistic way and the nondeterminism arises from the fact that we can have many different schedulers. We may now ask the question: how powerful are the syntactic schedulers wrt the semantic ones, i.e. those defined directly over the automaton?

Let P be a CCS$_p$ process and P_σ be the CCS$_\sigma$ process obtained from P by applying a linear labeling. We say that the semantic scheduler ζ of P is equivalent to the syntactic scheduler S of P_σ, written $\zeta \sim_P S$, iff the automata $etree(P, \zeta)$ and $P_\sigma \parallel S$ are probabilistically bisimilar in the sense of [5].

A scheduler S is *non-blocking* for a process P if it always schedules some transitions, except when P itself is blocked. Let $Sem(P)$ be the set of the semantic schedulers for the process P and $Syn(P_\sigma)$ be the set of the non-blocking syntactic schedulers for process P_σ. Then we can show that for all semantic schedulers of P we can create a equivalent syntactic one for P_σ.

Proposition 3. *Let P be a CCS process and let P_σ be a CCS$_\sigma$ process obtained by adding a linear labeling to P. Then $\forall \zeta \in Sem(P)\ \exists S \in Syn(P_\sigma) : \zeta \sim_P S$.*

To obtain this result the label test (**if-then-else**) is crucial: the scheduler uses it to find out the result of the probabilistic choice and adapt its behaviour accordingly (as the semantic scheduler is allowed to do). For example let $P = l{:}(l_1{:}a +_p l_2 : b) \mid (l_3{:}c + l_4{:}d)$. For this process, the scheduler $l.(\textbf{if } l_1 \textbf{ then } l_3.l_1 \textbf{ else } l_4.l_2)$ first performs the probabilistic choice. If the result is $l_1{:}a$ it performs c, a, otherwise it performs d, b. This is also the reason we need labels for 0, in case it is one of the operands of the probabilistic choice.

One would expect to obtain also the inverse of Proposition 3, showing the same expressive power for the two kinds of schedulers. We believe that this is indeed true, but it is technically more diffucult to state. The reason is that the simple translation we did from CCS$_p$ processes to CCS$_\sigma$, namely adding a linear labeling, might introduce choices that are not present in the original process. For example let $P = (a +_p a) \mid (c + d)$ and $P_\sigma = l{:}(l_1{:}a +_p l_2 : a) \mid (l_3{:}c + l_4{:}d)$. In P the choice $a +_p a$ is not a real choice, it can only do an τ transition and go to a with probability 1. But in P_σ we make the two outcomes distinct due to the labeling. So the syntactic scheduler $l.(\textbf{if } l_1 \textbf{ then } l_3.l_1 \textbf{ else } l_4.l_2)$ has no semantic counterpart simply because P_σ has more choices that P, but this is an artifact of the translation. A more precise translation that would establish the exact equivalence of schedulers is left as future work.

4.1 Using Non-linear Labelings

Up to now we are using only linear labelings which, as we saw, give us the whole power of semantic schedulers. However, we can construct non-linear labelings that are still deterministic, that is there is still only one transition possible at any time even though we have multiple occurrences of the same label. There are various cases of useful non-linear labelings.

Proposition 4. *Let P,Q be CCS$_\sigma$ processes with deterministic labelings (not necessarily disjoint). The following labelings are all deterministic:*

$$l\!:\!(P +_p Q) \tag{2}$$

$$l_1\!:\!a.P + l_2\!:\!b.Q \tag{3}$$

$$(\nu a)(\nu b)(l_1\!:\!a.P + l_1\!:\!b.Q \mid l_2\!:\!\bar{a}) \tag{4}$$

Consider the case where P and Q in the above proposition share the same labels. In (2) the scheduler cannot select an action inside P, Q, it must select the choice itself. After the choice, only one of P, Q will be available so there will be no ambiguity in selecting transitions. The case (3) is similar but with nondeterministic choice. Now the guarding prefixes must have different labels, since the scheduler should be able to resolve the choice, however after the choice only one of P, Q will be available. Hence, again, the multiple copies of the labels do not constitute a problem. In (4) we allow the same label on the guarding prefixes of a nondeterministic choice. This is because the guarding channels a, b are restricted and only one of the corresponding output actions is available ($\bar{a}$). As a consequence, there is no ambiguity in selecting transitions. A scheduler (l_1, l_2) can only perform a synchronization on a, even though l_1 appears twice.

However, using multiple copies of a label limits the power of the scheduler, since the labels provide information about the outcome of a probabilistic choice. In fact, this is exactly the technique we will use to achieve the goals described in the introduction. Consider for example the process $l\!:\!(l_1\!:\!\bar{a}.R_1 +_p l_1\!:\!\bar{a}.R_2) \mid l_2\!:\! a.P \mid l_3\!:\!a.Q$. From Proposition 4(2) its labeling is deterministic. However, since both branches of the probabilistic sum have the same label l_1, and the labels inside R_1, R_2 are not in the top-level so they cannot be used in a label test, the scheduler cannot resolve the choice between P and Q based on the outcome of the choice. There is still nondeterminism: the scheduler $l.(l_1, l_2)$ will select P and the scheduler $l.(l_1, l_3)$ will select Q. However this selection will be independent from the outcome of the probabilistic choice.

Note that we did not impose any direct restrictions on the schedulers, we still consider all possible syntactic schedulers for the process above. However, having the same label twice limits the power of the syntactic schedulers with respect to the semantic ones. This approach has the advantage that the restrictions are limited to the choices with the same label. We already know that having pairwise distinct labels gives the full power of the semantic scheduler. So the restriction is local to the place where we, intentionally, put the same labels.

5 Testing Relations for CCS$_\sigma$ Processes

Testing relations [19] are a method of comparing processes by considering their interaction with the environment. A *test* is a process running in parallel with the one being tested and which can perform a distinguished action ω that represents success. Two processes are testing equivalent if they can pass the same tests. This idea is very useful for the analysis of security protocols, as suggested in [20], since a test can be seen as an adversary who interferes with a communication agent and declares ω if an attack is successful.

In the probabilistic setting we take the approach of [13] which considers the exact probability of passing a test. This approach leads to the definition of two preorders $\sqsubseteq_{\mathbf{may}}$ and $\sqsubseteq_{\mathbf{must}}$. $P \sqsubseteq_{\mathbf{may}} Q$ means that if P can pass O then Q can also pass O with the same probability. $P \sqsubseteq_{\mathbf{must}} Q$ means that if P always passes O with at least some probability then Q always passes O with at least the same probability.

A labeling of a process is *fresh* (with respect to a set $\mathcal{P}$ of processes) if it is linear and its labels do not appear in any other process in $\mathcal{P}$. A test O is a CCS$_\sigma$ process with a fresh labeling, containing the distinguished action ω. Let $Test_{\mathcal{P}}$ denote the set of all tests with respect to $\mathcal{P}$ and let $(\nu)P$ denote the restriction on all channels of P, thus allowing only τ actions. We define $p_\omega(P,S,O)$ to be the probability of the set of executions of the fully probabilistic automaton $(\nu)(P \mid O) \parallel S$ that contain ω.

Definition 2. *Let P,Q be CCS$_\sigma$ processes. We define must and may testing preorders as follows:*

$$\begin{aligned} P \sqsubseteq_{\mathbf{may}} Q \quad &\text{iff } \forall O\ \forall S_P\ \exists S_Q\ :\ p_\omega(P,S_P,O) \le p_\omega(Q,S_Q,O) \\ P \sqsubseteq_{\mathbf{must}} Q \quad &\text{iff } \forall O\ \forall S_Q\ \exists S_P\ :\ p_\omega(P,S_P,O) \le p_\omega(Q,S_Q,O) \end{aligned}$$

where O ranges over $Test_{P,Q}$ and S_X ranges over $Syn((\nu)(X \mid O))$.

Also let $\approx_{\mathbf{may}}, \approx_{\mathbf{must}}$ be the equivalences induced by $\sqsubseteq_{\mathbf{may}}, \sqsubseteq_{\mathbf{must}}$ respectively.

A context C is a process with a hole. A preorder $\sqsubseteq$ is a precongruence if $P \sqsubseteq Q$ implies $C[P] \sqsubseteq C[Q]$ for all contexts C. May and must testing are precongruences if we restrict to contexts with fresh labelings and without occurrences of $+$. This is the analogous of the precongruence property in [3].

Proposition 5. *Let P,Q be CCS$_\sigma$ processes such that $P \sqsubseteq_{\mathbf{may}} Q$ and let C be a context with a fresh labeling and in which $+$ does not occur. Then $C[P] \sqsubseteq_{\mathbf{may}} C[Q]$. Similarly for $\sqsubseteq_{\mathbf{must}}$.*

This also implies that $\approx_{\mathbf{may}}, \approx_{\mathbf{must}}$ are congruences, except for $+$. The problem with $+$ is that P and $\tau.P$ are must equivalent, but $Q+P$ and $Q+\tau.P$ are not. This is typical for the CCS $+$: usually it does not preserve weak equivalences. Note that P,Q in the above proposition are not required to have linear labelings. This means that some choices inside P may be hidden from the scheduler while the context is fully visible, i.e. the scheduler's restriction is *local*.

If we remove the freshness condition then Proposition 5 is no longer true. Let $P = l_1 : a.l_2 : b$, $Q = l_3 : a.l_4 : b$ and $C = l : (l_1 : a.l_2 : c +_p [\,])$. We have $P \approx_{\mathbf{may}} Q$ but $C[P], C[Q]$ can be separated by the test $O = \bar{a}.\bar{b}.\omega \mid \bar{a}.\bar{c}.\omega$ (the labeling is omitted for simplicity since tests always have fresh labelings). It is easy to see that $C[Q]$ can pass the test with probability 1 by selecting the correct branch of O based on the outcome of the probabilistic choice. In $C[P]$ this is not possible because of the labels l_1, l_2 that are common in P, C.

We can now state formally the result that we announced in the introduction.

Theorem 1. *Let P, Q be CCS_σ processes and C a context with a fresh labeling and without occurrences of bang. Then*

$$l{:}(C[l_1{:}\tau.P] +_p C[l_1{:}\tau.Q]) \approx_{\textit{may}} C[l{:}(P +_p Q)] \quad \textit{and}$$
$$l{:}(C[l_1{:}\tau.P] +_p C[l_1{:}\tau.Q]) \approx_{\textit{must}} C[l{:}(P +_p Q)]$$

There are two crucial points in the above theorem. The first is that the labels of the context are replicated, thus the scheduler cannot use them to distinguish between $C[l_1 : \tau.P]$ and $C[l_1 : \tau.Q]$. The second is that P, Q are protected by a τ action labeled by the same label l_1. This is to ensure that in the case of a nondeterministic sum ($C = R + []$) the scheduler cannot find out whether the second operand of the choice is P or Q before it actually selects the second operand. For example let $R = a +_{0.5} 0$, $P = a$, $Q = 0$ (all omitted labels are fresh). Then $R_1 = (R + P) +_{0.1} (R + Q)$ is not testing equivalent to $R_2 = R + (P +_{0.1} Q)$ since they can be separated by $O = \overline{a}.\omega$ and a scheduler that resolves $R+P$ to P and $R+Q$ to R (it will be of the form **if** l_P **then** S_P **else** S_R). However, if we take $R'_1 = (R + l_1 : \tau.P) +_{0.1} (R + l_1 : \tau.Q)$ then R'_1 is testing equivalent to R_2 since now the scheduler cannot see the labels of P, Q so if it selects P then it is bound to also select Q.

The problem with bang is the persistence of the processes. Clearly $!P +_p !Q$ cannot be equivalent to $!(P +_p Q)$, since the first replicates only one of P, Q while the second replicates both. However Theorem 1 and Proposition 5 imply that $C'[l : (C[l_1 : \tau.P] +_p C[l_1 : \tau.Q])] \approx_{\mathbf{may}} C'[C[l : (P +_p Q)]]$, where C is a context without bang and C' is a context without $+$. The same is also true for $\approx_{\mathbf{must}}$. This means that we can lift the sum towards the root of the context until we reach a bang. Intuitively we cannot move the sum outside the bang since each replicated copy must perform a different probabilistic choice with a possibly different outcome.

6 An Application to Security

In this section we discuss an application of our framework to anonymity. In particular, we show how to specify the Dining Cryptographers protocol [21] so that it is robust to scheduler-based attacks. We first propose a method to encode *secret value passing*, which will turn out to be useful for the specification

6.1 Encoding Secret Value Passing

We propose to encode the passing of a secret message as follows:

$$l{:}c(x).P \;\triangleq\; \textstyle\sum_i l{:}cv_i.P[v_i/x] \qquad\qquad l{:}\bar{c}\langle v\rangle.P \;\triangleq\; l{:}\overline{cv}.P$$

This is the usual encoding of value passing in CSS except that we use the same label in all the branches of the nondeterministic sum. To ensure that the resulting labeling is deterministic we should restrict the channels cv_i and make sure that there is at most one output on c. We write $(\nu c)P$ for $(\nu cv_1) \ldots (\nu cv_n)P$. For

$$Master \triangleq l_1 : \textstyle\sum_{i=0}^{2} p_i (\underbrace{\overline{m}_0\langle i == 0\rangle}_{l_2} \mid \underbrace{\overline{m}_1\langle i == 1\rangle}_{l_3} \mid \underbrace{\overline{m}_2\langle i == 2\rangle}_{l_4})$$

$$Crypt_i \triangleq \underbrace{m_i(pay)}_{l_{5,i}} . \underbrace{c_{i,i}(coin_1)}_{l_{6,i}} . \underbrace{c_{i,i\oplus 1}(coin_2)}_{l_{7,i}} . \underbrace{\overline{out}_i\langle pay \otimes coin_1 \otimes coin_2\rangle}_{l_{8,i}}$$

$$Coin_i \triangleq l_{9,i} : ((\underbrace{\bar{c}_{i,i}\langle 0\rangle}_{l_{10,i}} \mid \underbrace{\bar{c}_{i\ominus 1,i}\langle 0\rangle}_{l_{11,i}}) +_{0.5} (\underbrace{\bar{c}_{i,i}\langle 1\rangle}_{l_{10,i}} \mid \underbrace{\bar{c}_{i\ominus 1,i}\langle 1\rangle}_{l_{11,i}}))$$

$$Prot \triangleq (\nu \boldsymbol{m})(Master \mid (\nu \boldsymbol{c})(\textstyle\prod_{i=0}^{2} Crypt_i \mid \prod_{i=0}^{2} Coin_i))$$

Fig. 5. Encoding of the dining cryptographers with probabilistic master

instance, the labeling of the process $(\nu c)(l_1 : c(x).P \mid l : (l_2 : \bar{c}\langle v_1\rangle +_p l_2 : \bar{c}\langle v_2\rangle))$ is deterministic. This example is indeed a combination of the cases (2) and (4) of Proposition 4. The two outputs on c are on different branches of the probabilistic sum, so during an execution at most one of them will be available. Thus there is no ambiguity in scheduling the sum produced by $c(x)$. The scheduler $l.(l_1, l_2)$ will perform a synchronization on cv_1 or cv_2, whatever is available after the probabilistic choice. Hence we have managed to hide the information about the value transmitted to P.

6.2 Dining Cryptographers with Probabilistic Master

The problem of the Dining Cryptographers is the following: Three cryptographers dine together. After the dinner, the bill has to be paid by either one of them or by another agent called the master. The master decides who will pay and then informs each of them separately whether he has to pay or not. The cryptographers would like to find out whether the payer is the master or one of them. However, in the latter case, they wish to keep the payer anonymous.

The Dining Cryptographers Protocol (DCP) solves the above problem as follows: each cryptographer tosses a fair coin which is visible to himself and his neighbor to the right. Each cryptographer checks the two adjacent coins and, if he is not paying, announces *agree* if they are the same and *disagree* otherwise. However, the paying cryptographer says the opposite. It can be proved that the master is paying if and only if the number of *disagrees* is even [21].

An external observer O is supposed to see only the three announcements $\overline{out}_i\langle \ldots \rangle$. As discussed in [22], DCP satisfies anonymity if we abstract from their order. If their order is observable, on the contrary, a scheduler can reveal the identity of the payer to O simply by forcing the payer to make his announcement first. Of course, this is possible only if the scheduler is unrestricted and can choose its strategy depending on the decision of the master or on the results of the coins.

In our framework we can solve the problem by giving a specification of the DCP in which the choices of the master and of the coins are made invisible to the scheduler. The specification is shown in Figure 5. The symbols $\oplus$ and $\ominus$ represent

$$P \ ::= \ldots \ \mid \ l{:}\{P\} \qquad\qquad \text{INDEP} \ \frac{P \parallel T \xrightarrow{\alpha} \mu}{l{:}\{P\} \parallel l.S, T \xrightarrow{\alpha} \mu'}$$
$$CP ::= P \parallel S, T \qquad\qquad \text{where } \mu'(P' \parallel S, T') = \mu(P' \parallel T')$$

Fig. 6. Adding an "independent" scheduler to the calculus

the addition and subtraction modulo 3, while $\otimes$ represents the addition modulo 2 (xor). The notation $i == n$ stands for 1 if $i = n$ and 0 otherwise.

There are many sources of nondeterminism: the order of communication between the master and the cryptographers, the order of reception of the coins, and the order of the announcements. The crucial points of our specification, which make the nondeterministic choices independent from the probabilistic ones, are: (a) all communications internal to the protocol are done by secret value passing, and (b) in each probabilistic choice the different branches have the same labels. For example, all branches of the master contain an output on m_0, always labeled by l_2, but with different values each time.

Thanks to the above independence, the specification satisfies strong probabilistic anonymity. There are various equivalent definitions of this property, we follow here the version presented in [22]. Let $\boldsymbol{o}$ represent an observable, i.e. a sequence of announcements, and $p_S(\boldsymbol{o} \mid \overline{m}_i\langle 1\rangle)$ the conditional probability, under scheduler S, that we get $\boldsymbol{o}$ given that Cryptographer i is the payer.

Proposition 6 (Strong probabilistic anonymity). *The protocol in Figure 5 satisfies the following property: for all schedulers S and for all observables $\boldsymbol{o}$: $p_S(\boldsymbol{o} \mid \overline{m}_0\langle 1\rangle) = p_S(\boldsymbol{o} \mid \overline{m}_1\langle 1\rangle) = p_S(\boldsymbol{o} \mid \overline{m}_2\langle 1\rangle)$.*

Note that different schedulers will produce different traces (we still have nondeterminism) but they will not depend on the choice of the master.

Some previous treatment of the DCP, including [22], solve the problem of the leak of information due to too-powerful schedulers by simply considering as observable sets of announcements instead than sequences. Thus one could think that using a true concurrent semantics, for instance event structures, would solve the problem in general. This is false: for instance, true concurrency would not help in the anonymity example in the introduction.

6.3 Dining Cryptographers with Nondeterministic Master

We sketch here a method to hide also certain nondeterministic choices from the scheduler, and we show an application to the variant of the Dining Cryptographers with nondeterministic master.

First we need to extend the calculus with a second *independent* scheduler T that we assume to resolve the nondeterministic choices that we want to make transparent to the main scheduler S. The new syntax and semantics are shown in Figure 6. $l : \{P\}$ represents a process where the scheduling of P is protected from the main scheduler S. The scheduler S can "ask" T to schedule P by selecting the label l. Then T resolves the nondeterminism of P as expressed by the INDEP rule. Note that we need to adjust also the other rules of the semantics to take T into account, but this change is straightforward. We assume that T does not collaborate with S so we do not need to worry about the labels in P.

To model the dining cryptographers with nondeterministic master we replace the *Master* process in Figure 5 by the following one.

$$Master \overset{\Delta}{=} l_1 : \{ \textstyle\sum_{i=0}^{2} l_{12,i} : \tau . (\underbrace{\overline{m}_0 \langle i == 0 \rangle}_{l_2} \mid \underbrace{\overline{m}_1 \langle i == 1 \rangle}_{l_3} \mid \underbrace{\overline{m}_2 \langle i == 2 \rangle}_{l_4}) \}$$

Essentially we have replaced the probabilistic choice by a *protected* nondeterministic one. Note that the labels of the operands are different but this is not a problem since this choice will be scheduled by T. Note also that after the choice we still have the same labels l_2, l_3, l_4, however the labeling is still deterministic.

In case of a nondeterministic selection of the culprit, and a probabilistic anonymity protocol, the notion of strong probabilistic anonymity has not been established yet, although some possible definitions have been discussed in [22]. Our framework makes it possible to give a natural and precise definition.

Definition 3 (Strong probabilistic anonymity for nondeterministic selection of the culprit). *A protocol with nondeterministic selection of the culprit satisfies strong probabilistic anonymity iff for all observables* $\boldsymbol{o}$*, schedulers* S*, and independent schedulers* T_1, T_2 *which select different culprits, we have:* $p_{S,T_1}(\boldsymbol{o}) = p_{S,T_2}(\boldsymbol{o})$.

Proposition 7. *The DCP with nondeterministic selection of the culprit specified in this section satisfies strong probabilistic anonymity.*

7 Conclusion and Future Work

We have proposed a process-calculus approach to the problem of limiting the power of the scheduler so that it does not reveal the outcome of hidden random choices, and we have shown its applications to the specification of information-hiding protocols. We have also discussed a feature, namely the distributivity of certain contexts over random choices, that makes our calculus appealing for verification. Finally, we have considered the probabilistic testing preorders and shown that they are precongruences in our calculus.

Our plans for future work are in two directions: (a) we would like to investigate the possibility of giving a game-theoretic characterization of our notion of scheduler, and (b) we would like to incorporate our ideas in some existing probabilistic model checker, for instance PRISM.

Acknowledgments. We would like to thank Vincent Danos for pointing out to us an attack to the Dining Cryptographers protocol based on the scheduler, which inspired this work. We also thank Roberto Segala and Daniele Varacca for their valuable comments on a previous version of this paper.

References

1. Vardi, M.: Automatic verification of probabilistic concurrent finite-state programs. In: Proc. of the Symp. on Foundations of Comp. Sci., pp. 327–338. IEEE Computer Society Press, Los Alamitos (1985)
2. Hansson, H., Jonsson, B.: A framework for reasoning about time and reliability. In: Proceedings of the Symp. on Real-Time Systems, pp. 102–111. IEEE Computer Society Press, Los Alamitos (1989)
3. Yi, W., Larsen, K.: Testing probabilistic and nondeterministic processes. In: Proc. of the IFIP Symp. on Protocol Specification, Testing and Verification (1992)
4. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, MIT/LCS/TR-676 (1995)
5. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing 2(2), 250–273 (1995)
6. Hansson, H., Jonsson, B.: A calculus for communicating systems with time and probabitilies. In: Proc. of the Real-Time Systems Symp., pp. 278–287. IEEE Computer Society Press, Los Alamitos (1990)
7. Bandini, E., Segala, R.: Axiomatizations for probabilistic bisimulation. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 370–381. Springer, Heidelberg (2001)
8. Andova, S.: Probabilistic process algebra. PhD thesis, TU Eindhoven (2002)
9. Mislove, M., Ouaknine, J., Worrell, J.: Axioms for probability and nondeterminism. In: Proc. of EXPRESS. ENTCS, vol. 96, pp. 7–28. Elsevier, Amsterdam (2004)
10. Palamidessi, C., Herescu, O.: A randomized encoding of the π-calculus with mixed choice. Theoretical Computer Science 335(2-3), 373–404 (2005)
11. Deng, Y., Palamidessi, C., Pang, J.: Compositional reasoning for probabilistic finite-state behaviors. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity. LNCS, vol. 3838, pp. 309–337. Springer, Heidelberg (2005)
12. Sokolova, A., de Vink, E.: Probabilistic automata: system types, parallel composition and comparison. In: Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P., Siegle, M. (eds.) Validation of Stochastic Systems. LNCS, vol. 2925, pp. 1–43. Springer, Heidelberg (2004)
13. Jonsson, B., Larsen, K., Yi, W.: Probabilistic extensions of process algebras. In: Handbook of Process Algebra, pp. 685–710. Elsevier, Amsterdam (2001)
14. Chatzikokolakis, K., Palamidessi, C.: A framework for analyzing probabilistic protocols and its application to the partial secrets exchange. In: De Nicola, R., Sangiorgi, D. (eds.) TGC 2005. LNCS, vol. 3705, pp. 146–162. Springer, Heidelberg (2005)
15. Canetti, R., Cheung, L., Kaynar, D., Liskov, M., Lynch, N., Pereira, O., Segala, R.: Task-structured probabilistic i/o automata. In: Proc. of WODES (2006)
16. Canetti, R., Cheung, L., Kaynar, D., Liskov, M., Lynch, N., Pereira, O., Segala, R.: Time-bounded task-PIOAs: A framework for analyzing security protocols. In: Dolev, S. (ed.) DISC 2006. LNCS, vol. 4167, pp. 238–253. Springer, Heidelberg (2006)
17. Garcia, F., van Rossum, P., Sokolova, A.: Probabilistic anonymity and admissible schedulers, arXiv:0706.1019v1 (2007)

18. de Alfaro, L., Henzinger, T., Jhala, R.: Compositional methods for probabilistic systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, Springer, Heidelberg (2001)
19. De Nicola, R., Hennessy, M.: Testing equivalences for processes. Theoretical Computer Science 34(1-2), 83–133 (1984)
20. Abadi, M., Gordon, A.: A calculus for cryptographic protocols: The spi calculus. Information and Computation 148(1), 1–70 (1999)
21. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology 1, 65–75 (1988)
22. Bhargava, M., Palamidessi, C.: Probabilistic anonymity. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 171–185. Springer, Heidelberg (2005)

Strategy Logic

Krishnendu Chatterjee[1], Thomas A. Henzinger[1,2], and Nir Piterman[2]

[1] University of California, Berkeley, USA
[2] EPFL, Switzerland
c_krish@eecs.berkeley.edu, {tah,Nir.Piterman}@epfl.ch

Abstract. We introduce *strategy logic*, a logic that treats strategies in two-player games as explicit first-order objects. The explicit treatment of strategies allows us to specify properties of nonzero-sum games in a simple and natural way. We show that the one-alternation fragment of strategy logic is strong enough to express the existence of Nash equilibria and secure equilibria, and subsumes other logics that were introduced to reason about games, such as ATL, ATL*, and game logic. We show that strategy logic is decidable, by constructing tree automata that recognize sets of strategies. While for the general logic, our decision procedure is nonelementary, for the simple fragment that is used above we show that the complexity is polynomial in the size of the game graph and optimal in the size of the formula (ranging from polynomial to 2EXPTIME depending on the form of the formula).

1 Introduction

In *graph games*, two players move a token across the edges of a graph in order to form an infinite path. The vertices are partitioned into player-1 and player-2 nodes, depending on which player chooses the successor node. The objective of player 1 is to ensure that the resulting infinite path lies inside a given winning set Ψ_1 of paths. If the game is zero-sum, then the goal of player 2 is to prevent this. More generally, in a nonzero-sum game, player 2 has her own winning set Ψ_2.

Zero-sum graph games have been widely used in the synthesis (or control) of reactive systems [22,24], as well as for defining and checking the realizability of specifications [1,8], the compatibility of interfaces [7], simulation relations between transition systems [11,19], and for generating test cases [3], to name just a few of their applications. The study of nonzero-sum graph games has been more recent, with assume-guarantee synthesis [4] as one of its applications.

The traditional formulation of graph games consists of a two-player graph (the "arena") and winning conditions Ψ_1 and Ψ_2 for the two players (in the zero-sum case, $\Psi_1 = \neg\Psi_2$), and asks for computing the winning sets W_1 and W_2 of vertices for the two players (in the zero-sum case, determinacy [18] ensures that $W_1 = \neg W_2$). To permit the unambiguous, concise, flexible, and structured expression of problems and solutions involving graph games, researchers have introduced *logics* that are interpreted over two-player graphs. An example is the temporal logic ATL [2], which replaces the unconstrained path quantifiers of CTL with constrained path quantifiers: while the CTL formula $\forall\Psi$ asserts that the path property Ψ is inevitable —i.e., Ψ holds on all paths from a given state— the ATL formula $\langle\langle 1\rangle\rangle\Psi$ asserts that Ψ is enforcible by player 1 —i.e., player 1 has a strategy so that Ψ holds on all paths that can result from playing that strategy. The logic ATL has proved useful for expressing proof obligations in system verification, as well as for expressing subroutines of verification algorithms.

However, because of limitations inherent in the definition of ATL, several extensions have been proposed [2], among them the temporal logic ATL*, the alternating-time μ-calculus, and a so-called *game logic* of [2]: these are motivated by expressing general ω-regular winning conditions, as well as tree properties of computation trees that result from fixing the strategy of one player (module checking [17]). All of these logics treat strategies implicitly through modalities. This is convenient for zero-sum games, but awkward for nonzero-sum games. Indeed, it was not known if Nash equilibria, one of the most fundamental concepts in game theory, can be expressed in these logics.

In order to systematically understand the expressiveness of game logics, and to specify nonzero-sum games, we study in this paper a logic that treats strategies as explicit first-order objects. For example, using explicit strategy quantifiers, the ATL formula $\langle\langle 1\rangle\rangle\Psi$ becomes $(\exists x \in \Sigma)(\forall y \in \Gamma)\Psi(x,y)$ —i.e., "there exists a player-1 strategy x such that for all player-2 strategies y, the unique infinite path that results from the two players following the strategies x and y satisfies the property Ψ." Strategies are a natural primitive when talking about games and winning, and besides ATL and its extensions, Nash equilibria are naturally expressible in *strategy logic*.

As an example, we define *winning secure equilibria* [5] in strategy logic. A winning secure equilibrium is a special kind of Nash equilibrium, which is important when reasoning about the components of a system, each with its own specification. At such an equilibrium, both players can collaborate to satisfy the combined objective $\Psi_1 \wedge \Psi_2$. Moreover, whenever player 2 decides to abandon the collaboration and enforce $\neg\Psi_1$, then player 1 has the ability to retaliate and enforce $\neg\Psi_2$; that is, player 1 has a winning strategy for the relativized objective $\Psi_2 \Rightarrow \Psi_1$ (where $\Rightarrow$ denotes implication). The symmetric condition holds for player 2; in summary: $(\exists x \in \Sigma)(\exists y \in \Gamma)[(\Psi_1 \wedge \Psi_2)(x,y) \ \wedge \ (\forall y' \in \Gamma)(\Psi_2 \Rightarrow \Psi_1)(x,y') \ \wedge \ (\forall x' \in \Sigma)(\Psi_1 \Rightarrow \Psi_2)(x',y)]$. Note that the same player-1 strategy x which is involved in producing the outcome $\Psi_1 \wedge \Psi_2$ must be able to win for $\Psi_2 \Rightarrow \Psi_1$; such a condition is difficult to state without explicit quantification over strategies.

Our results are twofold. First, we study the expressive power of strategy logic. We show that the logic is rich enough to express many interesting properties of zero-sum and nonzero-sum games that we know, including ATL*, game logic (and thus module checking), Nash equilibria, and secure equilibria. Indeed, ATL* and the equilibria can be expressed in a simple fragment of strategy logic with no more than one quantifier alternation (note the $\exists\forall$ alternation in the above formula for defining winning secure equilibria). We also show that the simple one-alternation fragment can be translated to ATL* (the translation in general is double exponential in the size of the formula) and thereby the equilibria can be expressed in ATL^*.

Second, we analyze the computational complexity of strategy logic. We show that, provided all winning conditions are specified in linear temporal logic (or by word automata), strategy logic is decidable. The proof goes through automata theory, using tree automata to specify the computation trees that result from fixing the strategy of one player. The complexity is nonelementary, with the number of exponentials depending on the quantifier alternation depth of the formula. In the case of the simple one-alternation fragment of strategy logic, which suffices to express ATL^* and equilibria, we obtain much better bounds: for example, for infinitary path formulas (path formulas that are independent of finite prefixes), there is a linear translation of a simple one-alternation fragment formula to an ATL^* formula.

In summary, strategy logic provides a decidable language for talking in a natural and uniform way about all kinds of properties on game graphs, including zero-sum, as well as nonzero-sum objectives. Of course, for more specific purposes, such as zero-sum reachability games, more restrictive and less expensive logics, such as ATL, are more appropriate; however, the consequences of such restrictions, and their relationships, is best studied within a clean, general framework such as the one provided by strategy logic. In other words, strategy logic can play for reasoning about games the same role that first-order logic with explicit quantification about time has played for temporal reasoning: the latter has been used to categorize and compare temporal logics (i.e., logics with implicit time), leading to a notion of completeness and other results in correspondence theory [10,15].

In this work we consider perfect-information games and, consequently, only pure strategies (no probabilistic choice). An extension of this work to the setting of partial-information games is an interesting research direction (cf. [12]). Other possible extensions include reasoning about concurrent games and about perfect-information games with probabilistic transitions, as well as increasing the expressive power of the logic by allowing more ways to bound strategies (e.g., comparing strategies).

2 Graph Games

A *game graph* $G = ((S, E), (S_1, S_2))$ consists of a directed graph (S, E) with a finite set S of states, a set E of edges, and a partition (S_1, S_2) of the state space S. The states in S_1 are called player-1 states; the states in S_2, player-2 states. For a state $s \in S$, we write $E(s)$ to denote the set $\{t \mid (s, t) \in E\}$ of successor states. We assume that every state has at least one out-going edge; i.e., $E(s)$ is nonempty for all $s \in S$.

Plays. A game is played by two players: player 1 and player 2, who form an infinite path in the game graph by moving a token along edges. They start by placing the token on an initial state and then they take moves indefinitely in the following way. If the token is on a state in S_1, then player 1 moves the token along one of the edges going out of the state. If the token is on a state in S_2, then player 2 does likewise. The result is an infinite path $\pi = \langle s_0, s_1, s_2, \ldots \rangle$ in the game graph; we refer to such infinite paths as plays. Hence given a game graph G, a *play* is an infinite sequence $\langle s_0, s_1, s_2, \ldots \rangle$ of states such that for all $k \geq 0$, we have $(s_k, s_{k+1}) \in E$. We write Π for the set of all plays.

Strategies. A strategy for a player is a recipe that specifies how to extend plays. Formally, a *strategy* σ for player 1 is a function σ: $S^* \cdot S_1 \to S$ that given a finite sequence of states, which represents the history of the play so far, and which ends in a player-1 state, chooses the next state. A strategy must choose only available successors, i.e., for all $w \in S^*$ and all $s \in S_1$, we have $\sigma(w \cdot s) \in E(s)$. The strategies for player 2 are defined symmetrically. We denote by Σ and Γ the sets of all strategies for player 1 and player 2, respectively. Given a starting state $s \in S$, a strategy σ for player 1, and a strategy τ for player 2, there is a unique play, denoted as $\pi(s, \sigma, \tau) = \langle s_0, s_1, s_2, \ldots \rangle$, which is defined as follows: $s = s_0$, and for all $k \geq 0$, we have (a) if $s_k \in S_1$, then $\sigma(s_0, s_1, \ldots, s_k) = s_{k+1}$, and (b) if $s_k \in S_2$, then $\tau(s_0, s_1, \ldots, s_k) = s_{k+1}$.

3 Strategy Logic

Strategy logic is interpreted over labeled game graphs. Let P be a finite set of atomic propositions. A *labeled game graph* $\mathcal{G} = (G, P, L)$ consists of a game graph G together with a labeling function L: $S \to 2^P$ that maps every state s to the set $L(s)$ of atomic propositions that are true at s. We assume that there is a special atomic proposition $\mathbf{tt} \in P$ such that $\mathbf{tt} \in L(s)$ for all $s \in S$.

Syntax. The formulas of strategy logic consist of the following kinds of subformulas. Path formulas Ψ are LTL formulas, which are interpreted over infinite paths of states. Atomic strategy formulas are path formulas $\Psi(x, y)$ with two arguments —a variable x that denotes a player-1 strategy, and a variable y that denotes a player-2 strategy. From atomic strategy formulas, we define a first-order logic of quantified strategy formulas. The formulas of strategy logic are the closed strategy formulas (i.e., strategy formulas without free strategy variables); they are interpreted over states. We denote path and strategy formulas by Ψ and Φ, respectively. We use the variables $x, x_1, x_2, \ldots$ to range over strategies for player 1, and denote the set of such variables by X; similarly, the variables $y, y_1, y_2, \ldots \in Y$ range over strategies for player 2. Formally, the path and strategy formulas are defined by the following grammar:

$$\Psi ::= p \mid \Phi \mid \Psi \wedge \Psi \mid \neg\Psi \mid \bigcirc\Psi \mid \Psi \,\mathcal{U}\, \Psi, \text{ where } p \in P \text{ and } \Phi \text{ is closed;}$$
$$\Phi ::= \Psi(x, y) \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid Qx.\Phi \mid Qy.\Phi, \text{ where } Q \in \{\exists, \forall\}, x \in X, y \in Y.$$

Observe that the closed strategy formulas can be reused as atomic propositions. We formally define the free variables of strategy formulas as follows:

$\mathsf{Free}(\Psi(x, y)) = \{x, y\}$;
$\mathsf{Free}(\Phi_1 \wedge \Phi_2) = \mathsf{Free}(\Phi_1) \cup \mathsf{Free}(\Phi_2)$;

$\mathsf{Free}(\Phi_1 \vee \Phi_2) = \mathsf{Free}(\Phi_1) \cup \mathsf{Free}(\Phi_2)$;
$\mathsf{Free}(Qx.\Phi') = \mathsf{Free}(\Phi') \setminus \{x\}$, for $Q \in \{\exists, \forall\}$;
$\mathsf{Free}(Qy.\Phi') = \mathsf{Free}(\Phi') \setminus \{y\}$, for $Q \in \{\exists, \forall\}$.

A strategy formula Φ is *closed* if $\mathsf{Free}(\Phi) = \emptyset$. We define additional boolean connectives such as $\Rightarrow$, and additional temporal operators such as $\Box$ and $\Diamond$, as usual.

Semantics. For a set $Z \subseteq X \cup Y$ of variables, a *strategy assignment* A_Z assigns to every variable $x \in Z \cap X$, a player-1 strategy $A_Z(x) \in \Sigma$, and to every variable $y \in Z \cap Y$, a player-2 strategy $A_Z(y) \in \Gamma$. Given a strategy assignment A_Z and player-1 strategy $\sigma \in \Sigma$, we denote by $A_Z[x \leftarrow \sigma]$ the extension of the assignment A_Z to the set $Z \cup \{x\}$, defined as follows: for $w \in Z \cup \{x\}$, we have $A_Z[x \leftarrow \sigma](w) = A_Z(w)$ if $w \neq x$, and $A_Z[x \leftarrow \sigma](x) = \sigma$. The definition of $A_Z[y \leftarrow \tau]$ for player-2 strategies $\tau \in \Gamma$ is analogous.

The semantics of path formulas Ψ is the usual semantics of LTL. We now describe the satisfaction of a strategy formula Φ at a state $s \in S$ with respect to a strategy assignment A_Z, where $\mathsf{Free}(\Phi) \subseteq Z$:

$$\begin{array}{lll}
(s, A_Z) \models \Psi(x,y) & \text{iff} & \pi(s, A_Z(x), A_Z(y)) \models \Psi; \\
(s, A_Z) \models \Phi_1 \wedge \Phi_2 & \text{iff} & (s, A_Z) \models \Phi_1 \text{ and } (s, A_Z) \models \Phi_2; \\
(s, A_Z) \models \Phi_1 \vee \Phi_2 & \text{iff} & (s, A_Z) \models \Phi_1 \text{ or } (s, A_Z) \models \Phi_2; \\
(s, A_Z) \models \exists x.\Phi' & \text{iff} & \exists \sigma \in \Sigma.\ (s, A_Z[x \leftarrow \sigma]) \models \Phi'; \\
(s, A_Z) \models \forall x.\Phi' & \text{iff} & \forall \sigma \in \Sigma.\ (s, A_Z[x \leftarrow \sigma]) \models \Phi'; \\
(s, A_Z) \models \exists y.\Phi' & \text{iff} & \exists \tau \in \Gamma.\ (s, A_Z[y \leftarrow \tau]) \models \Phi'; \\
(s, A_Z) \models \forall y.\Phi' & \text{iff} & \forall \tau \in \Gamma.\ (s, A_Z[y \leftarrow \tau]) \models \Phi'.
\end{array}$$

The semantics of a closed strategy formula Φ is the set $[\![\Phi]\!] = \{s \in S \mid (s, A_\emptyset) \models \Phi\}$ of states.

Unnested path formulas. Of special interest is the fragment of strategy logic where path formulas do not allow any nesting of temporal operators. This fragment has a CTL-like flavor, and as we show later, results in a decision procedure with a lower computational complexity. Formally, the *unnested* path formulas are restricted as follows:

$$\Psi ::= p \mid \Phi \mid \Psi \wedge \Psi \mid \neg\Psi \mid \bigcirc\Phi \mid \Phi\ \mathcal{U}\ \Phi, \text{ where } p \in P \text{ and } \Phi \text{ is closed.}$$

The resulting closed strategy formulas are called the *unnested-path-formula fragment* of strategy logic.

Examples. We now present some examples of formulas of strategy logic. We first show how to express formulas of the logics ATL and ATL^* [2] in strategy logic. The alternating-time temporal logic ATL^* consists of path formulas quantified by the alternating path operators $\langle\!\langle 1 \rangle\!\rangle$ and $\langle\!\langle 2 \rangle\!\rangle$, the existential path operator $\langle\!\langle 1, 2 \rangle\!\rangle$ (or $\exists$), and the universal path operator $\langle\!\langle \emptyset \rangle\!\rangle$ (or $\forall$). The logic ATL is the subclass of ATL^* where only unnested path formulas are considered. Some examples of ATL and ATL* formulas and the equivalent strategy formulas are as follows: for a proposition $p \in P$,

$$\langle\langle 1\rangle\rangle(\Diamond p) = \{s \in S \mid \exists\sigma.\ \forall\tau.\ \pi(s,\sigma,\tau) \models \Diamond p\} = [\![\exists x.\ \forall y.\ (\Diamond p)(x,y)]\!];$$
$$\langle\langle 2\rangle\rangle(\Box\Diamond p) = \{s \in S \mid \exists\tau.\ \forall\sigma.\ \pi(s,\sigma,\tau) \models \Box\Diamond p\} = [\![\exists y.\ \forall x.\ (\Box\Diamond p)(x,y)]\!];$$
$$\langle\langle 1,2\rangle\rangle(\Box p) = \{s \in S \mid \exists\sigma.\ \exists\tau.\ \pi(s,\sigma,\tau) \models \Box p\} = [\![\exists x.\ \exists y.\ (\Box p)(x,y)]\!];$$
$$\langle\langle \emptyset\rangle\rangle(\Diamond\Box p) = \{s \in S \mid \forall\sigma.\ \forall\tau.\ \pi(s,\sigma,\tau) \models \Box p\} = [\![\forall x.\ \forall y.\ (\Diamond\Box p)(x,y)]\!].$$

Consider the strategy formula $\Phi = \exists x.\ (\exists y_1.\ (\Box p)(x,y_1)\ \wedge\ \exists y_2.\ (\Box q)(x,y_2))$. This formula is different from the two formulas $\langle\langle 1,2\rangle\rangle(\Box p) \wedge \langle\langle 1,2\rangle\rangle(\Box q)$ (which is too weak) and $\langle\langle 1,2\rangle\rangle(\Box(p \wedge q))$ (which is too strong). It follows from the results of [2] that the formula Φ cannot be expressed in ATL*.

One of the features of strategy logic is that we can restrict the kinds of strategies that interest us. For example, the following strategy formula describes the states from which player 1 can ensure the goal Φ_1 while playing against any strategy that ensures Φ_2 for player 2:

$$\exists x_1.\ \forall y_1.\ ((\forall x_2.\Phi_2(x_2,y_1))\ \Rightarrow\ \Phi_1(x_1,y_1))$$

The mental exercise of "I know that you know that I know that you know ..." can be played in strategy logic up to any constant level. The analogue of the above formula, where the level of knowledge is nested up to level k, can be expressed in strategy logic. For example, the formula above ("knowledge nesting 1") is different from the following formula with "knowledge nesting 2":

$$\exists x_1.\ \forall y_1.\ ((\forall x_2.(\forall y_2.\Phi_1(x_2,y_2))\ \Rightarrow\ \Phi_2(x_2,y_1))\ \Rightarrow\ \Phi_1(x_1,y_1))$$

We do not know whether the corresponding fixpoint of 'full knowledge nesting' can be expressed in strategy logic.

As another example, we consider the notion of dominating and dominated strategies [21]. Given a path formula Ψ and a state $s \in S$, a strategy x_1 for player 1 *dominates* another player-1 strategy x_2 if for all player-2 strategies y, whenever $\pi(s,x_2,y) \models \Psi$, then $\pi(s,x_1,y) \models \Psi$. The strategy x_1 is *dominating* if it dominates every player-1 strategy x_2. The following strategy formula expresses that x_1 is a dominating strategy:

$$\forall x_2.\ \forall y.\ (\Psi(x_2,y)\ \Rightarrow\ \Psi(x_1,y))$$

Given a path formula Ψ and a state $s \in S$, a strategy x_1 for player 1 is *dominated* if there is a player-1 strategy x_2 such that (a) for all player-2 strategies y_1, if $\pi(s,x_1,y_1) \models \Psi$, then $\pi(s,x_2,y_1) \models \Psi$, and (b) for some player-2 strategy y_2, we have both $\pi(s,x_2,y_2) \models \Psi$ and $\pi(s,x_1,y_2) \not\models \Psi$. The following strategy formula expresses that x_1 is a dominated strategy:

$$\exists x_2.\ ((\forall y_1.\ \Psi(x_1,y_1)\ \Rightarrow\ \Psi(x_2,y_1))\ \wedge\ (\exists y_2.\ \Psi(x_2,y_2) \wedge \neg\Psi(x_1,y_2)))$$

The formulas for dominating and dominated strategies express properties about strategies and are not closed formulas.

4 Simple One-Alternation Fragment of Strategy Logic

In this section we define a subset of strategy logic. Intuitively, the alternation depth of a formula is the number of changes between $\exists$ and $\forall$ quantifiers (a formal definition is given in Section 6). The subset we consider here is a subset of the formulas that allow only one alternation of strategy quantifiers. We refer to this subset as the simple one-alternation fragment. We show later how several important concepts in nonzero-sum games can be captured in this fragment.

Syntax. We are interested in strategy formulas that depend on three path formulas: Ψ_1, Ψ_2, and Ψ_3. The strategy formulas in the simple one-alternation fragment assert that there exist player-1 and player-2 strategies that ensure Ψ_1 and Ψ_2, respectively, and at the same time cooperate to satisfy Ψ_3. Formally, the *simple one-alternation* strategy formulas are restricted as follows:

$$\Phi ::= \Phi \wedge \Phi \mid \neg\Phi \mid \exists x_1.\ \exists y_1.\ \forall x_2.\ \forall y_2.\ (\Psi_1(x_1, y_2) \wedge \Psi_2(x_2, y_1) \wedge \Psi_3(x_1, y_1)),$$

where $x_1, x_2 \in X$, and $y_1, y_2 \in Y$. The resulting closed strategy formulas are called the *simple one-alternation fragment* of strategy logic. Obviously, the formulas have a single quantifier alternation. We use the abbreviation $(\exists\, \Psi_1,\ \exists\, \Psi_2,\ \Psi_3)$ for simple one-alternation strategy formulas of the form $\exists x_1.\exists y_1.\ \forall x_2.\forall y_2.\ (\Psi_1(x_1, y_2) \wedge \Psi_2(x_2, y_1) \wedge \Psi_3(x_1, y_1))$.

Notation. For a path formula Ψ and a state s we define the set $\mathsf{Win}_1(s, \Psi) = \{\sigma \in \Sigma \mid \forall\tau \in \Gamma.\ \pi(s, \sigma, \tau) \models \Psi\}$ to denote the set of player-1 strategies that enforce Ψ against all player-2 strategies. We refer to the strategies in $\mathsf{Win}_1(s, \Psi)$ as the *winning* player-1 strategies for Ψ from s. Analogously, we define $\mathsf{Win}_2(s, \Psi) = \{\tau \in \Gamma \mid \forall\sigma \in \Sigma.\ \pi(s, \sigma, \tau) \models \Psi\}$ as the set of winning player-2 strategies for Ψ from s. Using the notation Win_1 and Win_2, the semantics of simple one-alternation strategy formulas can be written as follows: if $\Phi = (\exists\, \Psi_1,\ \exists\, \Psi_2,\ \Psi_3)$, then $[\![\Phi]\!] = \{s \in S \mid \exists\sigma \in \mathsf{Win}_1(s, \Psi_1).\ \exists\tau \in \mathsf{Win}_2(s, \Psi_2).\ \pi(s, \sigma, \tau) \models \Psi_3\}$.

5 Expressive Power of Strategy Logic

In this section we show that ATL^* and several concepts in nonzero-sum games can be expressed in the simple one-alternation fragment of strategy logic. We also show that *game logic*, which was introduced in [2] to express the module-checking problem [17], can be expressed in the one-alternation fragment of strategy logic (but not in the simple one-alternation fragment).

Expressing ATL^* and ATL. For every path formula Ψ, we have

$$\langle\!\langle 1 \rangle\!\rangle(\Psi) = \{s \in S \mid \exists\sigma.\ \forall\tau.\ \pi(s, \sigma, \tau) \models \Psi\} = [\![\exists x.\ \forall y.\ \Psi(x, y)]\!] = [\![(\exists\Psi, \exists\mathbf{tt}, \mathbf{tt})]\!];$$
$$\langle\!\langle 1, 2 \rangle\!\rangle(\Psi) = \{s \in S \mid \exists\sigma.\ \exists\tau.\ \pi(s, \sigma, \tau) \models \Psi\} = [\![\exists x.\ \exists y.\ \Psi(x, y)]\!] = [\![(\exists\mathbf{tt}, \exists\mathbf{tt}, \Psi)]\!].$$

The formulas $\langle\!\langle 2 \rangle\!\rangle(\Psi)$ and $\langle\!\langle \emptyset \rangle\!\rangle(\Psi)$ can be expressed similarly. Hence the logic ATL^* can be defined in the simple one-alternation fragment of strategy logic, and ATL can be defined in the simple one-alternation fragment with unnested path formulas.

Expressing Nash equilibria. In nonzero-sum games the input is a labeled game graph and two path formulas, which express the objectives of the two players. We define Nash equilibria [13] and show that their existence can be expressed in the simple one-alternation fragment of strategy logic.

Payoff profiles. Given a labeled game graph (G, P, L), two path formulas Ψ_1 and Ψ_2, strategies σ and τ for the two players, and a state $s \in S$, the *payoff* for player ℓ, where $\ell \in \{1, 2\}$, is defined as follows:

$$p_\ell(s, \sigma, \tau, \Psi_\ell) = \begin{cases} 1 & \text{if } \pi(s, \sigma, \tau) \models \Psi_\ell; \\ 0 & \text{otherwise.} \end{cases}$$

The *payoff profile* (p_1, p_2) consists of the payoffs $p_1 = p_1(s, \sigma, \tau, \Psi_1)$ and $p_2 = p_2(s, \sigma, \tau, \Psi_2)$ for player 1 and player 2.

Nash equilibria. A *strategy profile* (σ, τ) consists of strategies $\sigma \in \Sigma$ and $\tau \in \Gamma$ for the two players. Given a labeled game graph (G, P, L) and two path formulas Ψ_1 and Ψ_2, the strategy profile (σ^*, τ^*) is a *Nash equilibrium* at a state $s \in S$ if the following two conditions hold:

$$\begin{aligned} &(1)\ \forall \sigma \in \Sigma.\ p_1(s, \sigma, \tau^*, \Psi_1) \leq p_1(s, \sigma^*, \tau^*, \Psi_1); \\ &(2)\ \forall \tau \in \Gamma.\ p_2(s, \sigma^*, \tau, \Psi_2) \leq p_2(s, \sigma^*, \tau^*, \Psi_2). \end{aligned}$$

The state sets of the corresponding payoff profiles are defined as follows: for $i, j \in \{0, 1\}$, we have

$$\begin{aligned} NE(i, j) = \{s \in S \mid &\text{ there exists a Nash equilibrium } (\sigma^*, \tau^*) \text{ at } s \text{ such that} \\ &p_1(s, \sigma^*, \tau^*, \Psi_1) = i \text{ and } p_2(s, \sigma^*, \tau^*, \Psi_2) = j\}. \end{aligned}$$

Existence of Nash equilibria. We now define the state sets of the payoff profiles for Nash equilibria by simple one-alternation strategy formulas. The formulas are as follows:

$$\begin{aligned} NE(1,1) &= [\![(\exists \mathbf{tt},\ \exists \mathbf{tt},\ \Psi_1 \land \Psi_2)]\!]; \\ NE(0,0) &= [\![(\exists \neg\Psi_2,\ \exists \neg\Psi_1,\ \mathbf{tt})]\!]; \\ NE(1,0) &= \{s \in S \mid \exists \sigma.\ (\exists \tau.\ \pi(s, \sigma, \tau) \models \Psi_1\ \land\ \forall \tau'.\ \pi(s, \sigma, \tau') \models \neg\Psi_2)\} \\ &= [\![(\exists \neg\Psi_2,\ \exists \mathbf{tt},\ \Psi_1)]\!]; \\ NE(0,1) &= [\![(\exists \mathbf{tt},\ \exists \neg\Psi_1,\ \Psi_2)]\!]. \end{aligned}$$

Expressing secure equilibria. A notion of conditional competitiveness in nonzero-sum games was formalized by introducing secure equilibria [5]. We show that the existence of secure equilibria can be expressed in the simple one-alternation fragment of strategy logic.

Lexicographic ordering of payoff profiles. We define two lexicographic orderings $\preceq_1$ and $\preceq_2$ on payoff profiles. For two payoff profiles (p_1, p_2) and (p_1', p_2'), we have

$$(p_1, p_2) \preceq_1 (p'_1, p'_2) \text{ iff } (p_1 \le p'_1) \vee (p_1 = p'_1 \wedge p_2 \ge p'_2);$$
$$(p_1, p_2) \preceq_2 (p'_1, p'_2) \text{ iff } (p_2 \le p'_2) \vee (p_2 = p'_2 \wedge p_1 \ge p'_1).$$

Secure equilibria. A secure equilibrium is a Nash equilibrium with respect to the lexicographic preference orderings $\preceq_1$ and $\preceq_2$ on payoff profiles for the two players. Formally, given a labeled game graph (G, P, L) and two path formulas Ψ_1 and Ψ_2, a strategy profile (σ^*, τ^*) is a *secure equilibrium* at a state $s \in S$ if the following two conditions hold:

(1) $\forall\sigma \in \Sigma.\, (p_1(s, \sigma, \tau^*, \Psi_1), p_2(s, \sigma, \tau^*, \Psi_2)) \preceq_1 (p_1(s, \sigma^*, \tau^*, \Psi_1), p_2(s, \sigma^*, \tau^*, \Psi_2));$

(2) $\forall\tau \in \Gamma.\, (p_1(s, \sigma^*, \tau, \Psi_1), p_2(s, \sigma^*, \tau, \Psi_2)) \preceq_2 (p_1(s, \sigma^*, \tau^*, \Psi_1), p_2(s, \sigma^*, \tau^*, \Psi_2)).$

The state sets of the corresponding payoff profiles are defined as follows: for $i, j \in \{0, 1\}$, we have

$$SE(i, j) = \{s \in S \mid \text{there exists a secure equilibrium } (\sigma^*, \tau^*) \text{ at } s \text{ such that } p_1(s, \sigma^*, \tau^*, \Psi_1) = i \text{ and } p_2(s, \sigma^*, \tau^*, \Psi_2) = j\}.$$

It follows from the definitions that the sets $SE(i, j)$, for $i, j \in \{0, 1\}$, can be expressed in the one-alternation fragment (in the $\exists\forall$ fragment). The state sets of maximal payoff profiles for secure equilibria are defined as follows: for $i, j \in \{0, 1\}$, we have

$$MS(i, j) = \{s \in SE(i, j) \mid \text{if } s \in SE(i', j'), \text{ then } (i', j') \preceq_1 (i, j) \wedge (i', j') \preceq_2 (i, j)\}.$$

The following alternative characterizations of these sets are established in [5]:

$$MS(1, 0) = \{s \in S \mid \mathsf{Win}_1(s, \Psi_1 \wedge \neg\Psi_2) \neq \emptyset\};$$
$$MS(0, 1) = \{s \in S \mid \mathsf{Win}_2(s, \Psi_2 \wedge \neg\Psi_1) \neq \emptyset\};$$
$$MS(1, 1) = \{s \in S \mid \exists\sigma \in \mathsf{Win}_1(s, \Psi_2 \Rightarrow \Psi_1).\ \exists\tau \in \mathsf{Win}_2(s, \Psi_1 \Rightarrow \Psi_2).\ \pi(s, \sigma, \tau) \models \Psi_1 \wedge \Psi_2\};$$
$$MS(0, 0) = S \setminus (MS(1, 0) \cup MS(0, 1) \cup MS(1, 1)).$$

Existence of secure equilibria. From the alternative characterizations of the state sets of the maximal payoff profiles for secure equilibria, it follows that these sets can be defined by simple one-alternation strategy formulas. The formulas are as follows:

$$MS(1, 0) = [\![(\exists(\Psi_1 \wedge \neg\Psi_2),\ \exists\mathbf{tt},\ \mathbf{tt})]\!];$$
$$MS(0, 1) = [\![(\exists\mathbf{tt},\ \exists(\Psi_2 \wedge \neg\Psi_1),\ \mathbf{tt})]\!];$$
$$MS(1, 1) = [\![(\exists(\Psi_2 \Rightarrow \Psi_1),\ \exists(\Psi_1 \Rightarrow \Psi_2),\ \Psi_1 \wedge \Psi_2)]\!].$$

The set $MS(0, 0)$ can be obtained by complementing the disjunction of the three formulas for $MS(1, 0)$, $MS(0, 1)$, and $MS(1, 1)$.

Game logic and module checking. The syntax of *game logic* [2] is as follows. State formulas have the form $\exists\!\!\exists\{1\}.\ \theta$ or $\exists\!\!\exists\{2\}.\ \theta$, where θ is a tree formula.

Tree formulas are (a) state formulas, (b) boolean combinations of tree formulas, and (c) either $\exists\Psi$ or $\forall\Psi$, where Ψ is a path formula. Informally, the formula $\exists\{1\}.\,\theta$ is true at a state if there is a strategy σ for player 1 such that the tree formula θ is satisfied in the tree that is generated by fixing the strategy σ for player 1 (see [2] for details). Game logic can be defined in the one-alternation fragment of strategy logic (but not in the simple one-alternation fragment). The following example illustrates how to translate a state formula of game logic into a one-alternation strategy formula:

$$[\![\exists\{1\}.(\exists\Psi_1 \wedge \forall\Psi_2 \vee \forall\Psi_3)]\!] = [\![\exists x.\ (\exists y_1.\ \Psi_1(x, y_1) \wedge \forall y_2.\ \Psi_2(x, y_2) \vee \forall y_3.\ \Psi_3(x, y_3))]\!]$$

Consequently, the module-checking problem [17] can be expressed by one-alternation strategy formulas.

The following theorem compares the expressive power of strategy logic and its fragments with ATL^*, game logic, the alternating-time μ-calculus [2,16], and monadic second-order logic [23,26] (see [6] for proofs).

Theorem 1. *1. The expressiveness of the simple one-alternation fragment of strategy logic coincides with* ATL^**, and the one-alternation fragment of strategy logic is more expressive than* ATL^**.*
2. The one-alternation fragment of strategy logic is more expressive than game logic, and game logic is more expressive than the simple one-alternation fragment of strategy logic.
3. The alternating-time μ*-calculus is not as expressive as the alternation-free fragment of strategy logic, and strategy logic is not as expressive as the alternating-time* μ*-calculus.*
4. Monadic second order logic is more expressive than strategy logic.

6 Model Checking Strategy Logic

In this section we solve the model-checking problem for strategy logic. We encode strategies by using strategy trees. We reason about strategy trees using tree automata, making our solution similar to Rabin's usage of tree automata for solving the satisfiability problem of monadic second-order logic [23]. We give the necessary definitions and proceed with the algorithm.

Strategy trees and tree automata. Given a finite set Υ of directions, an *Υ-tree* is a set $T \subseteq \Upsilon^*$ such that if $x \cdot \upsilon \in T$, where $\upsilon \in \Upsilon$ and $x \in \Upsilon^*$, then also $x \in T$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. For every $\upsilon \in \Upsilon$ and $x \in T$, the node x is the *parent* of $x \cdot \upsilon$. Each node $x \neq \varepsilon$ of T has a *direction* in Υ. The direction of the root is the symbol $\perp$ (we assume that $\perp \notin \Upsilon$). The direction of a node $x \cdot \upsilon$ is υ. We denote by $dir(x)$ the direction of node x. An Υ-tree T is a *full infinite tree* if $T = \Upsilon^*$. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$, and for every $x \in \pi$ there exists a unique $\upsilon \in \Upsilon$ such that $x \cdot \upsilon \in \pi$.

Given two finite sets Υ and Λ, a *Λ-labeled Υ-tree* is a pair $\langle T, \rho\rangle$, where T is an Υ-tree, and ρ: $T \to \Lambda$ maps each node of T to a letter in Λ. When Υ and Λ

are not important or clear from the context, we call $\langle T, \rho \rangle$ a labeled tree. We say that an $((\Upsilon \cup \{\perp\}) \times \Lambda)$-labeled Υ-tree $\langle T, \rho \rangle$ is *Υ-exhaustive* if for every node $z \in T$, we have $\rho(z) \in \{dir(z)\} \times \Lambda$.

Consider a game graph $G = ((S, E), (S_1, S_2))$. For $\alpha \in \{1, 2\}$, a strategy σ: $S^* \cdot S_\alpha \to S$ can be encoded by an S-labeled S-tree $\langle S^*, \rho \rangle$ by setting $\sigma(v) = \rho(v)$ for every $v \in S^* \cdot S_\alpha$. Notice that σ may be encoded by many different trees. Indeed, for a node $v = s_0 \cdots s_n$ such that either $s_n \in S_{3-\alpha}$ or there exists some i such that $(s_i, s_{i+1}) \notin E$, the label $\rho(v)$ may be set arbitrarily. We may encode k different strategies by considering an S^k-labeled S-tree. Given a letter $\lambda \in S^k$, we denote by λ_i the projection of λ on its i-th coordinate. In this case, the i-th strategy is $\sigma_i(v) = \rho(v)_i$ for every $v \in S^* \cdot S_\alpha$. Notice that the different encoded strategies may belong to different players. We refer to such trees as *strategy trees*, and from now on, we may refer to a strategy as a tree $\langle S^*, \sigma \rangle$. In what follows we encode strategies by strategy trees. We construct tree automata that accept the strategy assignments that satisfy a given formula of strategy logic.

We use tree automata to reason about strategy trees. As we only use well-known results about such automata, we do not give a full formal definition, and refer the reader to [25]. Here, we use *alternating parity tree automata* (APTs). The language of an automaton is the set of labeled trees that it accepts. The size of an automaton is measured by the number of states, and the index, which is a measure of the complexity of the acceptance (parity) condition. The important qualities of automata that are needed for this paper are summarized in Theorem 2 below.

Theorem 2. *1. Given an* LTL *formula Ψ, we can construct an APT $\mathcal{A}_\Psi$ with $2^{O(|\Psi|)}$ states and index* 3 *such that $\mathcal{A}_\Psi$ accepts all labeled trees all of whose paths satisfy Ψ* [27].

2. *Given two APTs $\mathcal{A}_1$ and $\mathcal{A}_2$ with n_1 and n_2 states and indices k_1 and k_2, respectively, we can construct APTs for the conjunction and disjunction of $\mathcal{A}_1$ and $\mathcal{A}_2$ with $n_1 + n_2$ states and index* $\max(k_1, k_2)$. *We can also construct an APT for the complementary language of $\mathcal{A}_1$ with n_1 states and index k_1* [20].
3. *Given an APT $\mathcal{A}$ with n states and index k over the alphabet $\Lambda \times \Lambda'$, we can construct an APT $\mathcal{A}'$ that accepts a labeled tree over the alphabet Λ if some extension (or all extensions) of the labeling with labels from Λ' is accepted by $\mathcal{A}$. The number of states of $\mathcal{A}'$ is exponential in $n \cdot k$, and its index is linear in $n \cdot k$* [20].
4. *Given an APT $\mathcal{A}$ with n states and index k, we can check whether the language of $\mathcal{A}$ is empty or universal in time exponential in $n \cdot k$* [9,20].

Model-checking algorithm. The complexity of the model-checking algorithm for strategy formulas depends on the number of quantifier alternations of a formula. We now formally define the alternation depth of a closed strategy formula. The *alternation depth of a variable* of a closed strategy formula is the number of quantifier switches ($\exists\forall$ or $\forall\exists$) that bind the variable. The *alternation depth of a closed strategy formula* is the maximal alternation depth of a variable occurring in the formula.

Given a strategy formula $\varPhi$, we construct by induction on the structure of the formula a nondeterministic parity tree (NPT) automaton that accepts the set of strategy assignments that satisfy the formula. Without loss of generality, we assume that the variables in $X \cup Y$ are not reused; that is, in a closed strategy formula, there is a one-to-one and onto relation between the variables and the quantifiers.

Theorem 3. *Given a labeled game graph $\mathcal{G}$ and a closed strategy formula $\varPhi$ of alternation depth d, we can compute the set $[\![\varPhi]\!]$ of states in time proportional to d-EXPTIME in the size of $\mathcal{G}$, and $(d+1)$-EXPTIME in the size of $\varPhi$. If $\varPhi$ contains only unnested path formulas, then the complexity in the size of the formula reduces to d-EXPTIME.*

Proof. The case where closed strategy formula $\varPhi$ is used as a state formula in a larger formula $\varPhi'$, is solved by first computing the set of states satisfying $\varPhi$, adding this information to the labeled game graph $\mathcal{G}$, and then computing the set of states satisfying $\varPhi'$. In addition, if d is the alternation-depth of $\varPhi$ then $\varPhi$ is a boolean combination of closed strategy formulas of alternation depth at most d. Thus, it suffices to handle a closed strategy formula, and reduce the boolean reasoning to intersection, union, and complementation of the respective sets.

Consider a strategy formula $\varPhi$. Let $Z = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ be the set of variables used in $\varPhi$. Consider the alphabet S^{n+m} and an S^{n+m}-labeled S-tree σ. For a variable $v \in X \cup Y$, we denote by σ_v the strategy that stands in the location of variable v and for a set $Z' \subseteq Z$ we denote by $\sigma_{Z'}$ the set of strategies for the variables in Z'. We now describe how to construct an APT that accepts the set of strategy assignments that satisfy $\varPhi$. We build the APT by induction on the structure of the formula. For a subformula $\varPhi'$ we consider the following cases.

Case 1. $\varPhi' = \varPsi(x,y)$ —by Theorem 2 we can construct an APT $\mathcal{A}$ that accepts trees all of whose paths satisfy $\varPsi$. According to Theorem 2, $\mathcal{A}$ has $2^{O(|\varPsi|)}$ states.

Case 2. $\varPhi' = \varPhi_1 \wedge \varPhi_2$ —given APTs $\mathcal{A}_1$ and $\mathcal{A}_2$ that accept the set of strategy assignments that satisfy $\varPhi_1$ and $\varPhi_2$, respectively; we construct an APT $\mathcal{A}$ for the conjunction of $\mathcal{A}_1$ and $\mathcal{A}_2$. According to Theorem 2, $|\mathcal{A}| = |\mathcal{A}_1| + |\mathcal{A}_2|$ and the index of $\mathcal{A}$ is the maximum of the indices of $\mathcal{A}_1$ and $\mathcal{A}_2$.

Case 3. $\varPhi' = \exists x.\varPhi_1$ —given an APT $\mathcal{A}_1$ that accepts the set of strategy assignments that satisfy $\varPhi_1$ we do the following. According to Theorem 2, we can construct an APT $\mathcal{A}'$ that accepts a tree iff there exists a way to extend the labeling of the tree with a labeling for the strategy for x such that the extended tree is accepted by $\mathcal{A}_1$. The number of states of $\mathcal{A}'$ is exponential in $n \cdot k$ and its index is linear in $n \cdot k$. The cases where $\varPhi' = \exists y.\varPhi_1$, $\varPhi' = \forall x.\varPhi_1$, and $\varPhi' = \forall y.\varPhi_1$ are handled similarly.

We note that for a closed strategy formula $\varPhi$, the resulting automaton reads $S^{\emptyset}$-labeled S-trees. Thus, the input alphabet of the automaton has a single input letter and it only reads the structure of the S-tree.

The above construction starts with an automaton that is exponential in the size of a given LTL formula and incurs an additional exponent for every quantifier. In order to pay an exponent 'only' for every quantifier alternation, we have to use nondeterministic and universal automata, and maintain them in this form as long as possible. Nondeterministic automata are good for existential quantification, which comes to them for free, and universal automata are good for universal quantification. By careful analysis of the quantifier alternation hierarchy, we can choose to create automata of the right kind (nondeterministic or universal), and maintain them in this form under disjunctions and conjunctions. Then, the complexity is $d+1$ exponents in the size of the formula and d exponents in the size of the game.

Consider the case where only unnested path formulas are used. Then, given a path formula $\Psi(x,y)$, we construct an APT $\mathcal{A}$ that accepts trees all of whose paths satisfy Ψ. As $\Psi(x,y)$ does not use nesting of temporal operators, we can construct $\mathcal{A}$ with a linear number of states in the size of Ψ.[1] It follows that the total complexity is d exponents in the size of the formula and d exponents in the size of the game. Thus in the case of unnested path formulas one exponent can be removed. The exact details are omitted due to lack of space. ∎

One-alternation fragment. Since ATL* can be expressed in the simple one-Xalternation fragment of strategy logic, it follows that model checking simple one-alternation strategy formulas is 2EXPTIME-hard [2]. Also, since module checking can be expressed in the one-alternation fragment, it follows that model checking one-alternation strategy formulas with unnested path formulas is EXPTIME-hard [17]. These lower bounds together with Theorem 3 yield the following results.

Theorem 4. *Given a labeled game graph $\mathcal{G}$ and a closed one-alternation strategy formula Φ, the computation of $[\![\Phi]\!]$ is EXPTIME-complete in the size of $\mathcal{G}$, and 2EXPTIME-complete in the size of Φ. If Φ contains only unnested path formulas, then the complexity in the size of the formula is EXPTIME-complete.*

Model checking the simple one-alternation fragment. We now present a model-checking algorithm for the simple one-alternation fragment of strategy logic, with better complexity than the general algorithm. We first present a few notations.

Notation. For a labeled game graph $\mathcal{G}$ and a set $U \subseteq S$ of states, we denote by $\mathcal{G} \restriction U$ the restriction of the labeled game graph to the set U, and we use the notation only when for all states $u \in U$, we have $E(u) \cap U \neq \emptyset$; i.e., all states in U have a successor in U. A path formula Ψ is *infinitary* if the set of paths that satisfy Ψ is independent of all finite prefixes. The classical Büchi, coBüchi, parity, Rabin, Streett, and Müller conditions are all infinitary conditions. Every

[1] For a single temporal operator the number of states is constant, and boolean combinations between two automata may lead to an automaton whose size is the product of the sizes of the two automata. The number of multiplications is at most logarithmic in the size of the formula, resulting in a linear total number of states.

LTL objective on a labeled game graph can be reduced to an infinitary condition, such as a parity or Müller condition, on a modified game graph.

Lemma 1. *Let $\mathcal{G}$ be a labeled game graph, and let $\Phi = (\exists\ \Psi_1,\ \exists\ \Psi_2,\ \Psi_3)$ be a simple one-alternation strategy formula with path formulas Ψ_1, Ψ_2, and Ψ_3 such that Ψ_1 and Ψ_2 are infinitary. Let $W_1 = \langle\langle 1\rangle\rangle(\Psi_1)$ and $W_2 = \langle\langle 2\rangle\rangle(\Psi_2)$. Then $[\![\Phi]\!] = \langle\langle 1,2\rangle\rangle(\Psi_1 \wedge \Psi_2 \wedge \Psi_3)$ in the restricted graph $\mathcal{G} \upharpoonright (W_1 \cap W_2)$.*

Lemma 2. *Let $\mathcal{G}$ be a labeled game graph, and let $\Phi = (\exists\ \Psi_1,\ \exists\ \Psi_2,\ \Psi_3)$ be a simple one-alternation strategy formula with unnested path formulas Ψ_1, Ψ_2, and Ψ_3. Let $W_1 = \langle\langle 1\rangle\rangle(\Psi_1)$ and $W_2 = \langle\langle 2\rangle\rangle(\Psi_2)$. Then $[\![\Phi]\!] = \langle\langle 1,2\rangle\rangle(\Psi_1 \wedge \Psi_2 \wedge \Psi_3) \cap W_1 \cap W_2$.*

Theorem 5. *Let $\mathcal{G}$ be a labeled game graph with n states, and let $\Phi = (\exists\ \Psi_1,\ \exists\ \Psi_2,$ break$\Psi_3)$ be a simple one-alternation strategy formula.*

1. *We can compute the set $[\![\Phi]\!]$ of states in $n^{2^{O(|\Phi|)}} \cdot 2^{2^{O(|\Phi|\cdot\log|\Phi|)}}$ time; hence for formulas Φ of constant length the computation of $[\![\Phi]\!]$ is polynomial in the size of $\mathcal{G}$. The computation of $[\![\Phi]\!]$ is 2EXPTIME-complete in the size of Φ.*
2. *If Ψ_1, Ψ_2, and Ψ_3 are unnested path formulas, then there is a ATL* formula Φ' with unnested path formulas such that $|\Phi'| = O(|\Psi_1| + |\Psi_2| + |\Psi_3|)$ and $[\![\Phi]\!] = [\![\Phi']\!]$. Therefore $[\![\Phi]\!]$ can be computed in polynomial time.*

Theorem 5 follows from Lemmas 1 and 2 (see [6] for the proofs). We present some details only for part (1): given Ψ_1, Ψ_2, and Ψ_3 as parity conditions, from Lemma 1, it follows that $[\![(\exists\Psi_1,\ \exists\Psi_2, \Psi_3)]\!]$ can be computed by first solving two parity games, and then model checking a graph with a conjunction of parity conditions (i.e., a Streett condition). Since an LTL formula Ψ can be converted to an equivalent deterministic parity automaton with $2^{2^{O(|\Psi|\cdot\log|\Psi|)}}$ states and $2^{O(|\Psi|)}$ parities (by converting Ψ to a nondeterministic Büchi automaton, and then determinizing), applying an algorithm for solving parity games [14] and a polynomial-time algorithm for model checking Streett conditions, we obtain the desired upper bound. Observe that the model-checking complexity of the simple one-alternation fragment of strategy logic with unnested path formulas, as well as the program complexity of the simple one-alternation fragment (i.e., the complexity in terms of the game graph, for formulas of bounded size), are exponentially better than the corresponding complexities of the full one-alternation fragment.

Acknowledgements. This research was supported in part by the Swiss National Science Foundation and by the NSF grants CCR-0225610 and CCR-0234690.

References

1. Abadi, M., Lamport, L., Wolper, P.: Realizable and unrealizable concurrent program specifications. In: Ronchi Della Rocca, S., Ausiello, G., Dezani-Ciancaglini, M. (eds.) Automata, Languages and Programming. LNCS, vol. 372, pp. 1–17. Springer, Heidelberg (1989)

2. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM 49, 672–713 (2002)
3. Blass, A., Gurevich, Y., Nachmanson, L., Veanes, M.: Play to test. In: Grieskamp, W., Weise, C. (eds.) FATES 2005. LNCS, vol. 3997, pp. 32–46. Springer, Heidelberg (2006)
4. Chatterjee, K., Henzinger, T.A.: Assume guarantee synthesis. In: 30th TACAS. LNCS, vol. 4424, pp. 261–275. Springer, Heidelberg (2007)
5. Chatterjee, K., Henzinger, T.A., Jurdziński, M.: Games with secure equilibria. In: 19th LICS, pp. 160–169. IEEE Computer Society Press, Los Alamitos (2004)
6. Chatterjee, K., Henzinger, T.A., Piterman, N.: Strategy logic. Technical Report UCB/EECS-2007-78, UC Berkeley (2007)
7. de Alfaro, L., Henzinger, T.A.: Interface automata. In: 9th FASE, pp. 109–120. ACM Press, New York (2001)
8. Dill, D.L.: Trace theory for automatic hierarchical verification of speed independent circuits. MIT Press, Cambridge (1989)
9. Emerson, E.A., Jutla, C., Sistla, A.P.: On model-checking for fragments of μ-calculus. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 385–396. Springer, Heidelberg (1993)
10. Gabbay, D., Pnueli, A., Shelah, S., Stavi, J.: On the temporal analysis of fairness. In: 7th POPL, pp. 163–173. ACM Press, New York (1980)
11. Henzinger, T.A., Kupferman, O., Rajamani, S.: Fair simulation. Information and Computation 173(1), 64–81 (2002)
12. Kaiser, L.: Game quantification on automatic structures and hierarchical model checking games. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 411–425. Springer, Heidelberg (2006)
13. Nash Jr., J.F.: Equilibrium points in n-person games. Proceedings of the National Academy of Sciences 36, 48–49 (1950)
14. Jurdziński, M.: Small progress measures for solving parity games. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 290–301. Springer, Heidelberg (2000)
15. Kamp, J.A.W.: Tense Logic and the Theory of Order. PhD thesis, UCLA (1968)
16. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science 27, 333–354 (1983)
17. Kupferman, O., Vardi, M.Y., Wolper, P.: Module checking. Information and Computation 164, 322–344 (2001)
18. Martin, D.A.: Borel determinacy. Annals of Mathematics 65, 363–371 (1975)
19. Milner, R.: An algebraic definition of simulation between programs. In: 2nd IJCAI, pp. 481–489. British Computer Society (1971)
20. Muller, D.E., Schupp, P.E.: Alternating automata on infinite trees. Theoretical Computer Science 54, 267–276 (1987)
21. Owen, G.: Game Theory. Academic Press, London (1995)
22. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: 16th POPL, pp. 179–190. ACM Press, New York (1989)
23. Rabin, M.O.: Decidability of second order theories and automata on infinite trees. Transaction of the AMS 141, 1–35 (1969)
24. Ramadge, P.J.G., Wonham, W.M.: The control of discrete event systems. IEEE Transactions on Control Theory 77, 81–98 (1989)
25. Thomas, W.: On the synthesis of strategies in infinite games. In: Mayr, E.W., Puech, C. (eds.) STACS 95. LNCS, vol. 900, pp. 1–13. Springer, Heidelberg (1995)
26. Thomas, W.: Languages, automata, and logic. In: Handbook of Formal Languages. Beyond Words, vol. 3, ch. 7, pp. 389–455. Springer, Heidelberg (1997)
27. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1–37 (1994)

Solving Games Via Three-Valued Abstraction Refinement[⋆]

Luca de Alfaro and Pritam Roy

Computer Engineering Department
University of California, Santa Cruz, USA

Abstract. Games that model realistic systems can have very large state-spaces, making their direct solution difficult. We present a symbolic abstraction-refinement approach to the solution of two-player games. Given a property, an initial set of states, and a game representation, our approach starts by constructing a simple abstraction of the game, guided by the predicates present in the property and in the initial set. The abstraction is then refined, until it is possible to either prove, or disprove, the property over the initial states. Specifically, we evaluate the property on the abstract game in three-valued fashion, computing an over-approximation (the *may* states), and an under-approximation (the *must* states), of the states that satisfy the property. If this computation fails to yield a certain yes/no answer to the validity of the property on the initial states, our algorithm refines the abstraction by splitting *uncertain* abstract states (states that are may-states, but not must-states). The approach lends itself to an efficient symbolic implementation. We discuss the property required of the abstraction scheme in order to achieve convergence and termination of our technique. We present the results for reachability and safety properties, as well as for fully general ω-regular properties.

1 Introduction

Games provide a computational model that is widely used in applications ranging from controller design, to modular verification, to system design and analysis. The main obstacle to the practical application of games to design and control problems lies in very large state space of games modeling real-life problems. In system verification, one of the main methods for coping with large-size problems is *abstraction.* An abstraction is a simplification of the original system model. To be useful, an abstraction should contain sufficient detail to enable the derivation of the desired system properties, while being succinct enough to allow for efficient analysis. Finding an abstraction that is simultaneously informative and succinct is a difficult task, and the most successful approaches rely on the automated construction, and gradual refinement, of abstractions. Given a system and the property, a coarse initial abstraction is constructed: this initial abstraction typically preserves only the information about the system that is most immediately involved in the property, such as the values of the state variables mentioned in the property. This initial abstraction is then gradually, and automatically, refined, until

[⋆] This research was supported in part by the NSF grant CCR-0132780.

the property can be proved or disproved, in the case of a verification problem, or until the property can be analyzed to the desired level of accuracy, in case of a quantitative problem.

One of the most successful techniques for automated abstraction refinement is the technique of *counterexample-guided refinement,* or CEGAR [2,5,3]. According to this technique, given a system abstraction, we check whether the abstraction satisfies the property. If the answer is affirmative, we are done. Otherwise, the check yields an *abstract counterexample,* encoding a set of "suspect" system behaviors. The abstract counterexample is then further analyzed, either yielding a concrete counterexample (a proof that the property does not hold), or yielding a refined abstraction, in which that particular abstract counterexample is no longer present. The process continues until either a concrete counterexample is found, or until the property can be shown to hold (i.e., no abstract counterexamples are left). The appeal of CEGAR lies in the fact that it is a fully automatic technique, and that the abstraction is refined on-demand, in a property-driven fashion, adding just enough detail as is necessary to perform the analysis. The CEGAR technique has been extended to games in *counterexample-guided control* [12].

We propose here an alternative technique to CEGAR for refining game abstractions: namely, we propose to use *three-valued* analysis [16,17,9] in order to guide abstraction refinement for games. Our proposed technique works as follows. Given a game abstraction, we analyze it in three-valued fashion, computing the set of *must-win* states, which are known to satisfy the property, and the set of *never-win* states, which are known not to satisfy the property; the remaining states, for which the satisfaction is unknown, are called the *may-win* states. If this three-valued analysis yields the desired information (for example, showing the existence of an initial state with a given property), the analysis terminates. Otherwise, we refine the abstraction in a way that reduces the number of *may-win* states. The abstraction refinement proceeds in a property-dependent way. For *reachability* properties, where the goal is to reach a set of target states, we refine the abstraction at the may-must border, splitting a may-win abstract state into two parts, one of which is known to satisfy the property (and that will become a must-win state). For the dual case of *safety* properties, where the goal is to stay always in a set of "safe" states, the refinement occurs at the may-never border. We show that the proposed abstraction refinement scheme can be uniformly extended to games with parity objectives: this enables the solution of games with arbitrary ω-regular objectives, via automata-theoretic transformations [19].

Our proposed three-valued abstraction refinement technique can be implemented in fully symbolic fashion, and it can be applied to games with both finite and infinite state spaces. The technique terminates whenever the game has a finite *region algebra* (a partition of the state space) that is closed with respect to Boolean and controllable-predecessor operators [10]: this is the case for many important classes of games, among which timed games [13,8]. Furthermore, we show that the technique never performs unnecessary refinements: the final abstraction is never finer than a region algebra that suffices for proving the property.

In its aim of reducing the number of may-states, our technique is related to the three-valued abstraction refinement schemes proposed for CTL and transition systems in [16,17]. Differently from these approaches, however, we avoid the explicit construction of the tree-valued transition relation of the abstraction, relying instead on *may* and *must* versions of the controllable predecessor operators. Our approach provides precision and efficiency benefits. In fact, to retain full precision, the must-transitions of a three-valued model need to be represented as hyper-edges, rather than normal edges [17,9,18]; in turn, hyper-edges are computationally expensive both to derive and to represent. The may and must predecessor operators we use provide the same precision as the hyper-edges, without the associated computational penalty. For a similar reason, we show that our three-valued abstraction refinement technique for game analysis is superior to the CEGAR technique of [12], in the sense that it can prove a given property with an abstraction that never needs to be finer, and that can often be coarser. Again, the advantage is due to the fact that [12] represents player-1 moves in the abstract model via must-edges, rather than must hyper-edges. A final benefit of avoiding the explicit construction of the abstract model, relying instead on predecessor operators, is that the resulting technique is simpler to present, and simpler to implement.

2 Preliminary Definitions

A two-player game structure $G = \langle S, \lambda, \delta \rangle$ consists of:

- A state space S.
- A turn function $\lambda : S \to \{1, 2\}$, associating with each state $s \in S$ the player $\lambda(s)$ whose turn it is to play at the state. We write $\sim\!1 = 2$, $\sim\!2 = 1$, and we let $S_1 = \{s \in S \mid \lambda(s) = 1\}$ and $S_2 = \{s \in S \mid \lambda(s) = 2\}$.
- A transition function $\delta : S \mapsto 2^S \setminus \emptyset$, associating with every state $s \in S$ a non-empty set $\delta(s) \subseteq S$ of possible successors.

The game takes place over the state space S, and proceeds in an infinite sequence of rounds. At every round, from the current state $s \in S$, player $\lambda(s) \in \{1, 2\}$ chooses a successor state $s' \in \delta(s)$, and the game proceeds to s'. The infinite sequence of rounds gives rise to a *path* $\overline{s} \in S^\omega$: precisely, a *path* of G is an infinite sequence $\overline{s} = s_0, s_1, s_2, \ldots$ of states in S such that for all $k \geq 0$, we have $s_{k+1} \in \delta(s_k)$. We denote by Ω the set of all paths.

2.1 Game Objectives

An *objective* Φ for a game structure $G = \langle S, \lambda, \delta \rangle$ is a subset $\Phi \subseteq S^\omega$ of the sequences of states of G. A *game*(G,Φ) consists of a game structure G together with an objective Φ for a player. We consider winning objectives that consist in ω-regular conditions [19]; in particular, we will present algorithms for reachability, safety, and parity objectives. We often use linear-time temporal logic (LTL) notation [14] when defining objectives. Given a subset $T \subseteq S$ of states, the *reachability* objective $\Diamond T = \{s_0, s_1, s_2, \cdots \in S^\omega \mid \exists k \geq 0 . s_k \in T\}$ consists of all paths that reach T; the *safety* objective $\Box T = \{s_0, s_1, s_2, \cdots \in S^\omega \mid \forall k \geq 0 . s_k \in T\}$ consists of all paths that stay in T forever. We also consider *parity* objectives: the ability to solve games with parity objectives suffices for solving games with arbitrary LTL (or omega-regular) winning objectives [19]. A parity objective is specified via a partition $\langle B_0, B_1, \ldots, B_n \rangle$ of S, for some

$n \geq 0$. Given a path $\overline{s}$, let *Infi*$(\overline{s}) \subseteq S$ be the set of states that occur infinitely often along $\overline{s}$, and let $MaxCol(\overline{s}) = \max\{i \in \{0, \ldots, n\} \mid B_i \cap Infi(\overline{s}) \neq \emptyset\}$ be the index of the largest partition visited infinitely often along the path. Then, $\varphi_n = \{\overline{s} \in \Omega \mid MaxCol(\overline{s})$ is even$\}$.

2.2 Strategies and Winning States

A *strategy* for player $i \in \{1, 2\}$ in a game $G = \langle S, \lambda, \delta \rangle$ is a mapping $\pi_i : S^* \times S_i \mapsto S$ that associates with every nonempty finite sequence σ of states ending in S_i, representing the past history of the game, a successor state. We require that, for all $\sigma \in S^\omega$ and all $s \in S_i$, we have $\pi_i(\sigma s) \in \delta(s)$. An initial state $s_0 \in S$ and two strategies π_1, π_2 for players 1 and 2 uniquely determine a sequence of states *Outcome*$(s_0, \pi_1, \pi_2) = s_0, s_1, s_2, \ldots$, where for $k > 0$ we have $s_{k+1} = \pi_1(s_0, \ldots, s_k)$ if $s_k \in S_1$, and $s_{k+1} = \pi_2(s_0, \ldots, s_k)$ if $s_k \in S_2$.

Given an initial state s_0 and a winning objective $\Phi \subseteq S^\omega$ for player $i \in \{1, 2\}$, we say that state $s \in S$ is *winning* for player i if there is a player-i strategy π_i such that, for all player $\sim i$ strategies $\pi_{\sim i}$, we have *Outcome*$(s_0, \pi_1, \pi_2) \in \Phi$. We denote by $\langle i \rangle \Phi \subseteq S$ the set of winning states for player i for objective $\Phi \subseteq S^\omega$. A result by [11], as well as the determinacy result of [15], ensures that for all ω-regular goals Φ we have $\langle 1 \rangle \Phi = S \backslash \langle 2 \rangle \neg \Phi$, where $\neg \Phi = S \backslash \Phi$. Given a set $\theta \subseteq S$ of initial states, and a property $\Phi \subseteq S^\omega$, we will present algorithms for deciding whether $\theta \cap \langle i \rangle \Phi \neq \emptyset$ or, equivalently, whether $\theta \subseteq \langle i \rangle \Phi$, for $i \in \{1, 2\}$.

2.3 Game Abstractions

An *abstraction* V of a game structure $G = \langle S, \lambda, \delta \rangle$ consists of a set $V \subseteq 2^{2^S \backslash \emptyset}$ of *abstract states:* each abstract state $v \in V$ is a non-empty subset $v \subseteq S$ of concrete states. We require $\bigcup V = S$. For subsets $T \subseteq S$ and $U \subseteq V$, we write:

$$U\!\downarrow = \textstyle\bigcup_{u \in U} u \qquad T\!\uparrow_V^m = \{v \in V \mid v \cap T \neq \emptyset\} \qquad T\!\uparrow_V^M = \{v \in V \mid v \subseteq T\}$$

Thus, for a set $U \subseteq V$ of abstract states, $U\!\downarrow$ is the corresponding set of concrete states. For a set $T \subseteq S$ of concrete states, $T\!\uparrow_V^m$ and $T\!\uparrow_V^M$ are the set of abstract states that constitute over and under-approximations of the concrete set T. We say that the abstraction V of a state-space S is *precise* for a set $T \subseteq S$ of states if $T\!\uparrow_V^m = T\!\uparrow_V^M$.

2.4 Controllable Predecessor Operators

Two-player games with reachability, safety, or ω-regular winning conditions are commonly solved using *controllable predecessor operators.* We define the *player-1 controllable predecessor operator* $\mathrm{Cpre}_1 : 2^S \mapsto 2^S$ as follows, for all $X \subseteq S$ and $i \in \{1, 2\}$:

$$\mathrm{Cpre}_i(X) = \{s \in S_i \mid \delta(s) \cap X \neq \emptyset\} \cup \{s \in S_{\sim i} \mid \delta(s) \subseteq X\}. \tag{1}$$

Intuitively, for $i \in \{1, 2\}$, the set $\mathrm{Cpre}_i(X)$ consists of the states from which player i can force the game to X in one step. In order to allow the solution of games on the abstract state space V, we introduce abstract versions of $\mathrm{Cpre}_\cdot$. As multiple concrete states may correspond to the same abstract state, we cannot compute, on the abstract state space, a precise analogous of $\mathrm{Cpre}_\cdot$. Thus, for player $i \in \{1,2\}$, we define two abstract operators: the *may* operator $\mathrm{Cpre}_i^{V,m} : 2^V \mapsto 2^V$, which constitutes an over-approximation of Cpre_i, and the *must* operator $\mathrm{Cpre}_i^{V,M} : 2^V \mapsto 2^V$, which constitutes an under-approximation of Cpre_i [9]. We let, for $U \subseteq V$ and $i \in \{1,2\}$:

$$\mathrm{Cpre}_i^{V,m}(U) = \mathrm{Cpre}_i(U\!\downarrow)\!\uparrow_V^m \qquad \mathrm{Cpre}_i^{V,M}(U) = \mathrm{Cpre}_i(U\!\downarrow)\!\uparrow_V^M . \tag{2}$$

By the results of [9], we have the duality

$$\mathrm{Cpre}_i^{V,M}(U) = V \setminus \mathrm{Cpre}_{\sim i}^{V,m}(V \setminus U).$$

The fact that $\mathrm{Cpre}_\cdot^{V,m}$ and $\mathrm{Cpre}_\cdot^{V,M}$ are over and under-approximations of the concrete predecessor operator is made precise by the following observation: for all $U \subseteq V$ and $i \in \{1,2\}$, we have $\mathrm{Cpre}_i^{V,M}(U)\!\downarrow \subseteq \mathrm{Cpre}^i(U\!\downarrow) \subseteq \mathrm{Cpre}_i^{V,m}(U)\!\downarrow$.

2.5 μ-Calculus

We will express our algorithms for solving games on the abstract state space in μ-calculus notation [11]. Consider a function $\gamma : 2^V \mapsto 2^V$, monotone when 2^V is considered as a lattice with the usual subset ordering. We denote by $\mu Z.\gamma(Z)$ (resp. $\nu Z.\gamma(Z)$) the *least* (resp. *greatest*) *fixpoint* of γ, that is, the least (resp. greatest) set $Z \subseteq V$ such that $Z = \gamma(Z)$. As is well known, since V is finite, these fixpoints can be computed via Picard iteration: $\mu Z.\gamma(Z) = \lim_{n\to\infty} \gamma^n(\emptyset)$ and $\nu Z.\gamma(Z) = \lim_{n\to\infty} \gamma^n(V)$. In the solution of parity games we will make use of nested fixpoint operators, which can be evaluated by nested Picard iteration [11].

3 Reachability and Safety Games

We present our three-valued abstraction refinement technique by applying it first to the simplest games: reachability and safety games. It is convenient to present the arguments first for reachability games; the results for safety games are then obtained by duality.

3.1 Reachability Games

Our three-valued abstraction-refinement scheme for reachability proceeds as follows. We assume we are given a game $G = \langle S, \lambda, \delta\rangle$, together with an initial set $\theta \subseteq S$ and a final set $T \subseteq S$, and an abstraction V for G that is precise for θ and T. The question to be decided is: $\theta \cap \langle 1\rangle\Diamond T = \emptyset$?

The algorithm proceeds as follows. Using the may and must predecessor operators, we compute respectively the set W_1^m of *may-winning* abstract states, and the set W_1^M of *must-winning* abstract states. If $W_1^m \cap \theta\!\uparrow_V^m = \emptyset$, then the algorithm answers the question No; if $W_1^M \cap \theta\!\uparrow_V^M \neq \emptyset$, then the algorithm answers the question Yes. Otherwise, the algorithm picks an abstract state v such that

$$v \in (W_1^m \setminus W_1^M) \cap \mathrm{Cpre}_1^{V,m}(W_1^M). \tag{3}$$

Algorithm 1. 3-valued Abstraction Refinement for Reachability Games

Input: A concrete game structure $G = \langle S, \lambda, \delta \rangle$, a set of initial states $\theta \subseteq S$, a set of target states $T \subseteq S$, and an abstraction $V \subseteq 2^{2^S \setminus \emptyset}$ that is precise for θ and T.
Output: Yes if $\theta \cap \langle 1 \rangle \Diamond T \neq \emptyset$, and No otherwise.

```
1.  while true do
2.      $W_1^M := \mu Y.(T{\uparrow}_V^M \cup \mathrm{Cpre}_1^{V,M}(Y))$
3.      $W_1^m := \mu Y.(T{\uparrow}_V^m \cup \mathrm{Cpre}_1^{V,m}(Y))$
4.      if $W_1^m \cap \theta{\uparrow}_V^m = \emptyset$ then return No
5.      else if $W_1^M \cap \theta{\uparrow}_V^M \neq \emptyset$ then return Yes
6.      else
7.          choose $v \in (W_1^m \setminus W_1^M) \cap \mathrm{Cpre}_1^{V,m}(W_1^M)$
8.          let $v_1 := v \cap \mathrm{Cpre}_1(W_1^M{\downarrow})$ and $v_2 := v \setminus v_1$
9.          let $V := (V \setminus \{v\}) \cup \{v_1, v_2\}$
10.     end if
11. end while
```

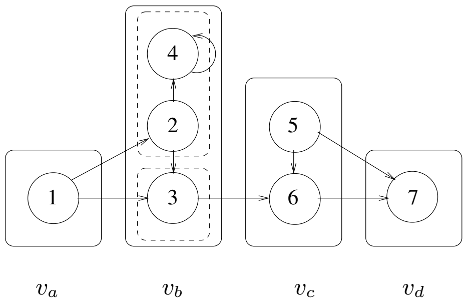


Fig. 1. Three-Valued Abstraction Refinement in Reachability Game

Such a state lies at the border between W_1^M and W_1^m. The state v is split into two abstract states v_1 and v_2, where:

$$v_1 = v \cap \mathrm{Cpre}_1(W_1^M{\downarrow}) \qquad\qquad v_2 = v \setminus \mathrm{Cpre}_1(W_1^M{\downarrow}).$$

As a consequence of (3), we have that $v_1, v_2 \neq \emptyset$. The algorithm is given in detail as Algorithm 1. We first state the partial correctness of the algorithm, postponing the analysis of its termination to Section 3.3.

Lemma 1. *At Step 4 of Algorithm 1, we have* $W_1^M{\downarrow} \subseteq \langle 1 \rangle \Diamond T \subseteq W_1^m{\downarrow}$.

Theorem 1. *(partial correctness) If Algorithm 1 terminates, it returns the correct answer.*

Example 1. As an example, consider the game G illustrated in Figure 1. The state space of the game is $S = \{1, 2, 3, 4, 5, 6, 7\}$, and the abstract state space is

Algorithm 2. 3-valued Abstraction Refinement for Safety Games

Input: A concrete game structure $G = \langle S, \lambda, \delta \rangle$, a set of initial states $\theta \subseteq S$, a set of target states $T \subseteq S$, and an abstraction $V \subseteq 2^{2^S \setminus \emptyset}$ that is precise for θ and T.
Output: Yes if $\theta \cap \langle 1 \rangle \Box T \neq \emptyset$, and No otherwise.

```
1.  while true do
2.      $W_1^M := \nu Y.(T\uparrow_V^M \cap \text{Cpre}_1^{V,M}(Y))$
3.      $W_1^m := \nu Y.(T\uparrow_V^m \cap \text{Cpre}_1^{V,m}(Y))$
4.      if $W_1^m \cap \theta\uparrow_V^m = \emptyset$ then return No
5.      else if $W_1^M \cap \theta\uparrow_V^M \neq \emptyset$ then return Yes
6.      else
7.          choose $v \in (W_1^m \setminus W_1^M) \cap \text{Cpre}_2^{V,m}(V \setminus W_1^m)$
8.          let $v_1 := v \cap \text{Cpre}_2(S \setminus (W_1^m\downarrow))$ and $v_2 := v \setminus v_1$
9.          let $V := (V \setminus \{v\}) \cup \{v_1, v_2\}$
10.     end if
11. end while
```

$V = \{v_a, v_b, v_c, v_d\}$, as indicated in the figure; the player-2 states are $S_2 = \{2, 3, 4\}$. We consider $\theta = \{1\}$ and $T = \{7\}$. After Steps 2 and 3 of Algorithm 1, we have $W_1^m = \{v_a, v_b, v_c, v_d\}$, and $W_1^M = \{v_c, v_d\}$. Therefore, the algorithm can answer neither No in Steps 4, nor Yes in Step 5, and proceeds to refine the abstraction. In Step 7, the only candidate for splitting is $v = v_b$, which is split into $v_1 = v_b \cap \text{Cpre}_1(W_1^M\downarrow) = \{3\}$, and $v_2 = v_b \setminus v_1 = \{2, 4\}$. It is easy to see that at the next iteration of the analysis, v_1 and v_a are added to W_1^M, and the algorithm returns the answer Yes. ∎

3.2 Safety Games

We next consider a safety game specified by a target $T \subseteq S$, together with an initial condition $\theta \subseteq S$. Given an abstraction V that is precise for T and θ, the goal is to answer the question of whether $\theta \cap \langle 1 \rangle \Box T = \emptyset$. As for reachability games, we begin by computing the set W_1^m of may-winning states, and the set W_1^M of must-winning states. Again, if $W_1^m \cap \theta\uparrow_V^m = \emptyset$, we answer No, and if $W_1^M \cap \theta\uparrow_V^M \neq \emptyset$, we answer Yes. In safety games, unlike in reachability games, we cannot split abstract states at the may-must boundary. For reachability games, a may-state can only win by reaching the goal T, which is contained in $W_1^M\downarrow$: hence, we refine the may-must border. In a safety game with objective $\Box T$, on the other hand, we have $W_1^m\downarrow \subseteq T$, and a state in $W_1^m\downarrow$ can be winning even if it never reaches $W_1^M\downarrow$ (which indeed can be empty if the abstraction is too coarse). Thus, to solve safety games, we split abstract states at the may-losing boundary, that is, at the boundary between W_1^m and its complement. This can be explained by the fact that $\langle 1 \rangle \Box T = S \setminus \langle 2 \rangle \Diamond \neg T$: the objectives $\Box T$ and $\Diamond \neg T$ are dual. Therefore, we adopt for $\Box T$ the same refinement method we would adopt for $\Diamond \neg T$, and the may-must boundary for $\langle 2 \rangle \Diamond \neg T$ is the may-losing boundary for $\langle 1 \rangle \Box T$. The algorithm is given below.

Lemma 2. *At Step 4 of Algorithm 2, we have* $W_1^M\downarrow \subseteq \langle 1 \rangle \Box T \subseteq W_1^m\downarrow$.

Theorem 2. *(partial correctness) If Algorithm 2 terminates, it returns the correct answer.*

3.3 Termination

We present a condition that ensures termination of Algorithms 1 and 2. The condition states that, if there is a finite algebra of regions (sets of concrete states) that is closed under Boolean operations and controllable predecessor operators, and that is precise for the sets of initial and target states, then (i) Algorithms 1 and 2 terminate, and (ii) the algorithms never produce abstract states that are finer than the regions of the algebra (guaranteeing that the algorithms do not perform unnecessary work). Formally, a *region algebra* for a game $G = \langle S, \lambda, \delta \rangle$ is an abstraction U such that:

- U is closed under Boolean operations: for all $u_1, u_2 \in U$, we have $u_1 \cup u_2 \in U$ and $S \setminus u_1 \in U$.
- U is closed under controllable predecessor operators: for all $u \in U$, we have $\text{Cpre}_1(u) \in U$ and $\text{Cpre}_2(u) \in U$.

Theorem 3. *(termination) Consider a game G with a finite region algebra U. Assume that Algorithm 1 or 2 are called with arguments G, θ, T, with $\theta, T \in U$, and with an initial abstraction $V \subseteq U$. Then, the following assertions hold for both algorithms:*

1. *The algorithms, during their executions, produce abstract states that are all members of the algebra U.*
2. *The algorithms terminate.*

The proof of the results is immediate. Many games, including timed games, have the finite region algebras mentioned in the above theorem [13,10].

3.4 Approximate Abstraction Refinement Schemes

While the abstraction refinement scheme above is fairly general, it makes two assumptions that may not hold in a practical implementation:

- it assumes that we can compute $\text{Cpre}_*^{V,m}$ and $\text{Cpre}_*^{V,M}$ of (2) precisely;
- it assumes that, once we pick an abstract state v to split, we can split it into v_1 and v_2 precisely, as outlined in Algorithms 1 and 2.

In fact, both assumptions can be related, yielding a more widely applicable abstraction refinement algorithm for two-player games. We present the modified algorithm for the reachability case only; the results can be easily extended to the dual case of safety objectives. Our starting point consists in approximate versions $\text{Cpre}_i^{V,m+}, \text{Cpre}_i^{V,M-} : 2^V \mapsto 2^V$ of the operators $\text{Cpre}_i^{V,m}$, $\text{Cpre}_i^{V,M}$, for $i \in \{1, 2\}$. We require that, for all $U \subseteq V$ and $i \in \{1, 2\}$, we have:

$$\text{Cpre}_i^{V,m}(U) \subseteq \text{Cpre}_i^{V,m+}(U) \qquad \text{Cpre}_i^{V,M-}(U) \subseteq \text{Cpre}_i^{V,M}(U) \; . \tag{4}$$

With these operators, we can phrase a new, approximate abstraction scheme for reachability, given in Algorithm 3. The use of the approximate operators means that, in Step 8,

Algorithm 3. Approximate 3-valued Abstraction Refinement for Reachability Games

Input: A concrete game structure $G = \langle S, \lambda, \delta \rangle$, a set of initial states $\theta \subseteq S$, a set of target states $T \subseteq S$, and an abstraction $V \subseteq 2^{2^S \setminus \emptyset}$ that is precise for θ and T.
Output: Yes if $\theta \cap \langle 1 \rangle \Diamond T \neq \emptyset$, and No otherwise.

```
1.  while true do
2.      $W_1^{M-} := \mu Y.(T{\uparrow}_V^M \cup \text{Cpre}_1^{V,M-}(Y))$
3.      $W_1^{m+} := \mu Y.(T{\uparrow}_V^m \cup \text{Cpre}_1^{V,m+}(Y))$
4.      if $W_1^{m+} \cap \theta{\uparrow}_V^m = \emptyset$ then return No
5.      else if $W_1^{M-} \cap \theta{\uparrow}_V^M \neq \emptyset$ then return Yes
6.      else
7.          choose $v \in (W_1^{m+} \setminus W_1^{M-}) \cap \text{Cpre}_1^{V,m+}(W_1^{M-})$
8.          let $v_1 := v \cap \text{Cpre}_1(W_1^{M-}{\downarrow})$
9.          if $v_1 = \emptyset$ or $v_1 = v$
10.             then split $v$ arbitrarily into non-empty $v_1$ and $v_2$
11.             else $v_2 = r \setminus v_1$
12.         end if
13.         let $V := (V \setminus \{v\}) \cup \{v_1, v_2\}$
14.     end if
15. end while
```

we can be no longer sure that both $v_1 \neq \emptyset$ and $v \setminus v_1 \neq \emptyset$. If the "precise" split of Step 8 fails, we resort instead to an arbitrary split (Step 10). The following theorem states that the algorithm essentially enjoys the same properties of the "precise" Algorithms 1 and 2.

Theorem 4. *The following assertions hold.*

1. Correctness. *If Algorithm 3 terminates, it returns the correct answer.*
2. Termination. *Assume that Algorithm 3 is given as input a game G with a finite region algebra U, and arguments $\theta, T \in U$, as well as with an initial abstraction $V \subseteq U$. Assume also that the region algebra U is closed with respect to the operators $\text{Cpre}_i^{V,M-}$ and $\text{Cpre}_i^{V,m+}$, for $i \in \{1, 2\}$, and that Step 10 of Algorithm 3 splits the abstract states in regions in U. Then, Algorithm 3 terminates, and it produces only abstract states in U in the course of its execution.*

3.5 Comparision with Counterexample-Guided Control

It is instructive to compare our three-valued refinement approach with the *counterexample-guided control* approach of [12]. In [12], an abstact game structure is constructed and analyzed. The abstract game contains *must* transitions for player 1, and *may* transitions for player 2. Every counterexample to the property (spoiling strategy for player 2) found in the abstract game is analyzed in the concrete game. If the counterexample is real, the property is disproved; If the counterexample is spurious, it is ruled out by refining the abstraction. The process continues until either the property is disproved, or no abstract counterexamples is found, proving the property.

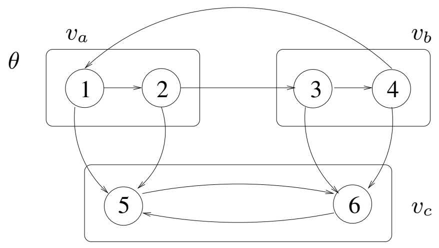


Fig. 2. Safety game, with objective $\Box T$ for $T = \{1, 2, 3, 4\}$

The main advantage of our proposed three-valued approach over counterexample-guided control is, somewhat paradoxically, that we do not explicitly construct the abstract game. It was shown in [17,9] that, for a game abstraction to be fully precise, the *must* transitions should be represented as hyper-edges (an expensive representation, space-wise). In the counterexample-guided approach, instead, normal *must* edges are used: the abstract game representation incurs a loss of precision, and more abstraction refinement steps may be needed than with our proposed three-valued approach. This is best illustrated with an example.

Example 2. Consider the game structure depicted in Figure 2. The state space is $S = \{1, 2, 3, 4, 5, 6\}$, with $S_1 = \{1, 2, 3, 4\}$ and $S_2 = \{5, 6\}$; the initial states are $\theta = \{1, 2\}$. We consider the safety objective $\Box T$ for $T = \{1, 2, 3, 4\}$. We construct the abstraction $V = \{v_a, v_b, v_c\}$ precise for θ and T, as depicted. In the counterexample-guided control approach of [12], hyper-must transitions are not considered in the construction of the abstract model, and the transitions between v_a and v_b are lost: the only transitions from v_a and v_b lead to v_c. Therefore, there is a spurious abstract counterexample tree $v_a \to v_c$; ruling it out requires splitting v_a into its constituent states 1 and 2. Once this is done, there is another spurious abstract counterexample $2 \to v_b \to v_c$; ruling it out requires splitting v_b in its constituent states. In contrast, in our approach we have immediately $W_1^M = \{v_a, v_b\}$ and $v_a, v_b \in \text{Cpre}_1^{V,M}(\{v_a, v_b\})$, so that no abstraction refinement is required. ∎

The above example illustrates that the counterexample-guided control approach of [12] may require a finer abstraction than our three-valued refinement approach, to prove a given property. On the other hand, it is easy to see that if an abstraction suffices to prove a property in the counterexample-guided control approach, it also suffices in our three-valued approach: the absence of abstract counterexamples translates directly in the fact that the states of interest are must-winning.

4 Symbolic Implementation

We now present a concrete symbolic implementation of our abstraction scheme. We chose a simple symbolic representation for two-player games; while the symbolic game representations encountered in real verification systems (see, e.g.,[6,7]) are usually more complex, the same principles apply.

4.1 Symbolic Game Structures

To simplify the presentation, we assume that all variables are Boolean. For a set X of Boolean variables, we denote by $\mathcal{F}(X)$ the set of propositional formulas constructed from the variables in X, the constants *true* and *false*, and the propositional connectives $\neg, \wedge, \vee, \rightarrow$. We denote with $\phi[\psi/x]$ the result of replacing all occurrences of the variable x in ϕ with a formula ψ. For $\phi \in \mathcal{F}(X)$ and $x \in X$, we write $\left\{\substack{\forall \\ \exists}\right\} x.\phi$ for $\phi[\textit{true}/x]\left\{\substack{\wedge \\ \vee}\right\}\phi[\textit{false}/x]$. We extend this notation to sets $Y = \{y_1, y_2, \ldots, y_n\}$ of variables, writing $\forall Y.\phi$ for $\forall y_1.\forall y_2.\cdots\forall y_n.\phi$, and similarly for $\exists Y.\phi$. For a set X of variables, we also denote by $X' = \{x' \mid x \in X\}$ the corresponding set of *primed* variables; for $\phi \in \mathcal{F}(X)$, we denote ϕ' the formula obtained by replacing every $x \in X$ with x'.

A *state* s over a set X of variables is a truth-assignment $s : X \mapsto \{\mathrm{T}, \mathrm{F}\}$ for the variables in X; we denote with $S[X]$ the set of all such truth assignments. Given $\phi \in \mathcal{F}(X)$ and $s \in S[X]$, we write $s \models \phi$ if ϕ holds when the variables in X are interpreted as prescribed by s, and we let $[\![\phi]\!]_X = \{s \in S[X] \mid s \models \phi\}$. Given $\phi \in \mathcal{F}(X \cup X')$ and $s, t \in S[X]$, we write $(s,t) \models \phi$ if ϕ holds when $x \in X$ has value $s(x)$, and $x' \in X'$ has value $t(x)$. When X, and thus the state space $S[X]$, are clear from the context, we equate informally formulas and sets of states. These formulas, or sets of states, can be manipulated with the help of symbolic representations such as BDDs [4]. A *symbolic game structure* $G_S = \langle X, \Lambda_1, \Delta\rangle$ consists of the following components:

– A set of Boolean variables X.
– A predicate $\Lambda_1 \in \mathcal{F}(X)$ defining when it is player 1's turn to play. We define $\Lambda_2 = \neg\Lambda_1$.
– A transition function $\Delta \in \mathcal{F}(X \cup X')$, such that for all $s \in S[X]$, there is some $t \in S[X]$ such that $(s,t) \models \Delta$.

A symbolic game structure $G_S = \langle X, \Lambda_1, \Delta\rangle$ induces a (concrete) game structure $G = \langle S, \lambda, \delta\rangle$ via $S = S[X]$, and for $s, t \in S$, $\lambda(s) = 1$ iff $s \models \Lambda_1$, and $t \in \delta(s)$ iff $(s,t) \models \Delta$. Given a formula $\phi \in \mathcal{F}(X)$, we have

$$\mathrm{Cpre}_1([\![\phi]\!]_X) = [\![\big(\Lambda_1 \wedge \exists X'.(\Delta \wedge \phi')\big) \vee \big(\neg\Lambda_1 \wedge \forall X'.(\Delta \rightarrow \phi')\big)]\!]_X.$$

4.2 Symbolic Abstractions

We specify an abstraction for a symbolic game structure $G_S = \langle X, \Lambda_1, \Delta\rangle$ via a subset $X^a \subseteq X$ of its variables: the idea is that the abstraction keeps track only of the values of the variables in X^a; we denote by $X^c = X \setminus X^a$ the concrete-only variables. We assume that $\Lambda_1 \in \mathcal{F}(X^a)$, so that in each abstract state, only one of the two players can move (in other words, we consider *turn-preserving* abstractions [9]). With slight abuse of notation, we identify the abstract state space V with $S[X^a]$, where, for $s \in S[X]$ and $v \in V$, we let $s \in v$ iff $s(x) = v(x)$ for all $x \in X^a$. On this abstract state space, the operators $\mathrm{Cpre}_1^{V,m}$ and $\mathrm{Cpre}_1^{V,M}$ can be computed symbolically via the corresponding operators $\mathrm{SCpre}_1^{V,m}$ and $\mathrm{SCpre}_1^{V,M}$, defined as follows. For $\phi \in \mathcal{F}(X^a)$,

$$\mathrm{SCpre}_1^{V,m}(\phi) = \exists X^c.\Big(\big(\Lambda_1 \wedge \exists X'.(\Delta \wedge \phi')\big) \vee \big(\Lambda_2 \wedge \forall X'.(\Delta \rightarrow \phi')\big)\Big) \qquad (5)$$

$$\mathrm{SCpre}_1^{V,M}(\phi) = \forall X^c.\Big(\big(\Lambda_1 \wedge \exists X'.(\Delta \wedge \phi')\big) \vee \big(\Lambda_2 \wedge \forall X'.(\Delta \rightarrow \phi')\big)\Big) \qquad (6)$$

The above operators correspond exactly to (2). Alternatively, we can abstract the transition formula $\varDelta$, defining:

$$\varDelta^m_{X^a} = \exists X^c . \exists X^{c\prime} . \varDelta \qquad \varDelta^M_{X^a} = \forall X^c . \exists X^{c\prime} . \varDelta \; .$$

These abstract transition relations can be used to compute approximate versions $\mathrm{SCpre}_1^{V,m+}$ and $\mathrm{SCpre}_1^{V,M-}$ of the controllable predecessor operators of (5), (6):

$$\mathrm{SCpre}_1^{V,m+}(\phi) = \Big(\big(\varLambda_1 \wedge \exists X^{a\prime}.(\varDelta^m_{X^a} \wedge \phi')\big) \vee \big(\varLambda_2 \wedge \forall X^{a\prime}.(\varDelta^m_{X^a} \rightarrow \phi')\big) \Big)$$
$$\mathrm{SCpre}_1^{V,M-}(\phi) = \Big(\big(\varLambda_1 \wedge \exists X^{a\prime}.(\varDelta^M_{X^a} \wedge \phi')\big) \vee \big(\varLambda_2 \wedge \forall X^{a\prime}.(\varDelta^M_{X^a} \rightarrow \phi')\big) \Big)$$

These operators, while approximate, satisfy the conditions (4), and can thus be used to implement symbolically Algorithm 3.

4.3 Symbolic Abstraction Refinement

We replace the abstraction refinement step of Algorithms 1, 2, and 3 with a step that adds a variable $x \in X^c$ to the set X^a of variables present in the abstraction. The challenge is to choose a variable x that increases the precision of the abstraction in a useful way. To this end, we follow an approach inspired directly by [5].

Denote by $v \in S[X^a]$ the abstract state that Algorithms 3 chooses for splitting at Step 7, and let $\psi_1^{M-} \in \mathcal{F}(X^a)$ be the formula defining the set W_1^{M-} in the same algorithm. We choose $x \in X^c$ so that there are at least two states $s_1, s_2 \in v$ that differ only for the value of x, and such that $s_1 \models \mathrm{SCpre}_1^{V,m+}(\psi_1^{M-})$ and $s_2 \not\models \mathrm{SCpre}_1^{V,m+}(\psi_1^{M-})$. Thus, the symbolic abstraction refinement algorithm first searches for a variable $x \in X^c$ for which the following formula is true:

$$\exists (X^c \backslash x) . \Big(\Big(\chi_v \rightarrow \big(x \equiv \mathrm{SCpre}_1^{V,m+}(\psi_1^{M-}) \big) \Big) \vee \Big(\chi_v \rightarrow \big(x \not\equiv \mathrm{SCpre}_1^{V,m+}(\psi_1^{M-}) \big) \Big) \Big),$$

where χ_v is the *characteristic formula* of v:

$$\chi_v = \bigwedge \{ x \mid x \in X^a . v(x) = \mathrm{T} \} \;\wedge\; \bigwedge \{ \neg x \mid x \in X^a . v(x) = \mathrm{F} \} \; .$$

If no such variable can be found, due to the approximate computation of $\mathrm{SCpre}_1^{V,m+}$ and $\mathrm{SCpre}_1^{V,M-}$, then $x \in X^c$ is chosen arbitrarily. The choice of variable for Algorithm 2 can be obtained by reasoning in dual fashion.

5 Abstraction Refinement for Parity Games

We now present a general abstraction-refinement algorithm to solve a n-color parity game where the state-space S is partitioned into n disjoint subsets $B_0, B_1, \ldots, B_n$. Denoting the parity condition $\langle B_0, \ldots, B_n \rangle$ by φ, the winning states can be computed as follows [11]:

$$\langle 1 \rangle \varphi = \varUpsilon_n Y_n . \ldots \nu Y_0 . \big((B_0 \cap \mathrm{Cpre}_1(Y_0)) \cup \ldots \cup (B_n \cap \mathrm{Cpre}_1(Y_n)) \big),$$

Algorithm 4. 3-valued Abstraction Refinement for Parity Games

Input: A concrete game structure $G = \langle S, \lambda, \delta \rangle$, a set of initial states $\theta \subseteq S$, a parity condition $\varphi = \langle B_0, B_1, \dots B_n \rangle$, and an abstraction $V \subseteq 2^{2^S \setminus \emptyset}$ that is precise for $\theta, B_0, \dots, B_n$.
Output: Yes if $\theta \cap \langle 1 \rangle \varphi \neq \emptyset$, and No otherwise.

```
1.  while true do
2.      $W_1^M := \mathit{Win}(\mathrm{Cpre}_1^{V,M}, \emptyset, n)$
3.      $W_1^m := \mathit{Win}(\mathrm{Cpre}_1^{V,m}, \emptyset, n)$
4.      if $W_1^m \cap \theta\uparrow_V^m = \emptyset$ then return No
5.      else if $W_1^M \cap \theta\uparrow_V^M \neq \emptyset$ then return Yes
6.      else
7.          choose $(v, v_1, v_2)$ from Split$(V, W_1^m, W_1^M, n)$
11.         $V = (V \setminus \{v\}) \cup \{v_1, v_2\}$
13.     end if
14. end while
```

where Υ_i is ν when i is even, and is μ when i is odd, for $i \in \mathbb{N}$. Algorithm 4 describes our 3-valued abstraction-refinement approach to solving parity games. The algorithm starts with an abstraction V that is precise for $B_0, \dots, B_n$. The algorithm computes the sets W_1^m and W_1^M using the following formula:

$$\mathit{Win}(\mathrm{Op}, U, k) = \Upsilon_k Y_k. \dots \nu Y_0. \begin{pmatrix} U \cup \ \left(B_k\uparrow_V^M \cap \ \mathrm{Op}(Y_k)\right) \\ \cup \ \cdots \\ \cup \ \left(B_0\uparrow_V^M \cap \ \mathrm{Op}(Y_0)\right) \end{pmatrix}.$$

In the formula, $U \subseteq V$ is a set of abstract states that are already known to be must-winning; in Algorithm 4 we use this formula with $\mathrm{Op} = \mathrm{Cpre}_i^{V,M}$ to compute W_1^M, and with $\mathrm{Op} = \mathrm{Cpre}_i^{V,m}$ to compute W_1^m.

The refinement step relies on a recursive function *Split* (Algorithm 5) to obtain a list of candidate splits (v, v_1, v_2): each of these suggests to split v into non-empty v_1 and v_2. The function *Split* is called with a partial parity condition $B_0, \dots, B_k$, for $0 \leq k \leq n$. The function first computes a candidate split in the color B_k: if k is even (resp. odd), it proceeds as in Steps 7–8 of Algorithm 2 (resp. Algorithm 1). The function then recursively computes the may-winning set of states in a game with $k - 1$ colors, where the states in U_M are already known to be must-winning, and computes additional candidate splits in such $k - 1$ color game. We illustrate the function *Split* with the help of an example.

Example 3. Figure 3 shows how function *Split* (Algorithm 5) computes the candidate splits in a Co-Büchi game with colors B_0, B_1 (the objective of player 1 consists in eventually forever staying in B_0). The candidate splits in B_1 are given by:

$$P_1 = \{(r, r_1, r_2) \mid r \in B_1 \cap (\ W_1^m \ \setminus \ W_1^M\ ) \cap \ \mathrm{Cpre}_1^{V,m}(W_1^M), \\ r_1 = r \cap \mathrm{Cpre}_1(W_1^M), r_2 = r \setminus r_1\}$$

To compute the candidate splits in B_0, the algorithm considers a safety game with goal $\Box B_0$, with W_1^M as set of states that are already considered to be winning; the

Algorithm 5. Split(V, U_m, U_M, k)

Input: An abstraction V, may winning set $U_m \subseteq V$, must winning set $U_M \subseteq V$, number of colors k.
Output: A set of tuples $(v, v_1, v_2) \in V \times 2^S \times 2^S$.

```
1.  if $k$ odd:
2.      then $P = \{(v, v_1, v_2) \mid v \in \{B_k \cap (U_m \setminus U_M) \cap \mathrm{Cpre}_1^{V,m}(U_M)\},$
              $v_1 = v \cap \mathrm{Cpre}_1(U_M), v_2 = v \setminus v_1, v_1 \neq \emptyset, v_2 \neq \emptyset\}$
3.      else $P = \{(v, v_1, v_2) \mid v \in \{B_k \cap (U_m \setminus U_M) \cap \mathrm{Cpre}_2^{V,m}(V \setminus U_m)\},$
              $v_1 = v \cap \mathrm{Cpre}_2(V \setminus U_m), v_2 = v \setminus v_1, v_1 \neq \emptyset, v_2 \neq \emptyset\}$
4.  end if
5.  if $k = 0$ then return $P$
6.  else
7.      $W_1^m := Win(\mathrm{Cpre}_1^{V,m}, U_M, k-1)$
8.      return $P \cup$ Split($V, W_1^m, U_M, k-1$))
9.  end if
```

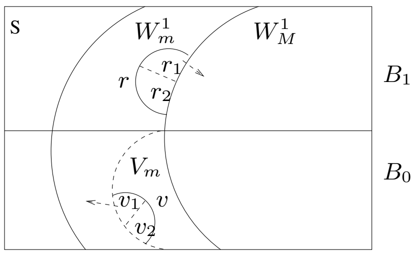


Fig. 3. Abstraction refinement for co-Büchi games

may-winning states in this game are $V_m = \nu Y_0.\ (W_1^M \cup (B_0 \cap \mathrm{Cpre}_1^{V,m}(Y_0)))$. Thus, the algorithm computes the following candidate splits in B_0:

$$P_0 = \{(v, v_1, v_2) \mid v \in B_0 \cap (V_m \setminus W_1^M) \cap \mathrm{Cpre}_2^{V,m}(V \setminus V_m),$$
$$v_1 = v \cap \mathrm{Cpre}_2(V \setminus V_m), v_2 = v \setminus v_1\}.$$

The function Split returns $P_1 \cup P_0$ as the set of candidate splits for the given co-Büchi game. ∎

Lemma 3. *At Step 4 of Algorithm 4, we have $W_1^M{\downarrow} \subseteq \langle 1 \rangle \varphi \subseteq W_1^m{\downarrow}$.*

Theorem 5. *If Algorithm 4 terminates, it returns the correct answer. Moreover, consider a game G with a finite region algebra U. Assume that Algorithm 4 is called with an initial abstraction $V \subseteq U$. Then, the algorithms terminates, and during its execution, it produces abstract states that are all members of the algebra U.*

6 Conclusion and Future Work

We have presented a technique for the verification of game properties based on the construction, three-valued analysis, and refinement of game abstractions. The approach is suitable for symbolic implementation and, being based entirely on the evaluation of predecessor operators, is simple both to present and to implement. We plan to implement the approach as part of the Ticc toolset of interface composition and analysis [1], applying it both to the untimed interface composition problem (which requires solving safety games), and to the timed interface composition problem (which requires solving 3-color parity games).

References

1. Adler, B., de Alfaro, L., Silva, L.D.D., Faella, M., Legay, A., Raman, V., Roy, P.: TICC: a tool for interface compatibility and composition. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 59–62. Springer, Heidelberg (2006)
2. Alur, R., Itai, A., Kurshan, R.P., Yannakakis, M.: Timing verification by successive approximation. Inf. Comput. 118(1), 142–157 (1995)
3. Ball, T., Rajamani, S.: The SLAM project: Debugging system software via static analysis. In: Proceedings of the 29th Annual Symposium on Principles of Programming Languages, pp. 1–3. ACM Press, New York (2002)
4. Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers C-35(8), 677–691 (1986)
5. Clarke, E., Grumberg, O., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, Springer, Heidelberg (2000)
6. de Alfaro, L., Alur, R., Grosu, R., Henzinger, T., Kang, M., Majumdar, R., Mang, F., Meyer-Kirsch, C., Wang, B.: Mocha: A model checking tool that exploits design structure. In: ICSE 01. Proceedings of the 23rd International Conference on Software Engineering, pp. 835–836 (2001)
7. de Alfaro, L., da Silva, L.D., Faella, M., Legay, A., Roy, P., Sorea, M.: Sociable interfaces. In: Gramlich, B. (ed.) Frontiers of Combining Systems. LNCS (LNAI), vol. 3717, pp. 81–105. Springer, Heidelberg (2005)
8. de Alfaro, L., Faella, M., Henzinger, T., Majumdar, R., Stoelinga, M.: The element of surprise in timed games. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 144–158. Springer, Heidelberg (2003)
9. de Alfaro, L., Godefroid, P., Jagadeesan, R.: Three-valued abstractions of games: Uncertainty, but with precision. In: Proc. 19th IEEE Symp. Logic in Comp. Sci., pp. 170–179. IEEE Computer Society Press, Los Alamitos (2004)
10. de Alfaro, L., Henzinger, T., Majumdar, R.: Symbolic algorithms for infinite-state games. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, Springer, Heidelberg (2001)
11. Emerson, E., Jutla, C.: Tree automata, mu-calculus and determinacy (extended abstract). In: Proc. 32nd IEEE Symp. Found. of Comp. Sci., pp. 368–377. IEEE Computer Society Press, Los Alamitos (1991)
12. Henzinger, T., Jhala, R., Majumdar, R.: Counterexample-guided control. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 886–902. Springer, Heidelberg (2003)

13. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems. In: Mayr, E.W., Puech, C. (eds.) STACS 95. LNCS, vol. 900, pp. 229–242. Springer, Heidelberg (1995)
14. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, New York (1991)
15. Martin, D.: An extension of Borel determinacy. Annals of Pure and Applied Logic 49, 279–293 (1990)
16. Shoham, S.: A game-based framework for CTL counter-examples and 3-valued abstraction-refinement. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 275–287. Springer, Heidelberg (2003)
17. Shoham, S., Grumberg, O.: Monotonic abstraction-refinement for CTL. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 546–560. Springer, Heidelberg (2004)
18. Shoham, S., Grumberg, O.: 3-valued abstraction: More precision at less cost. In: Proc. 21st IEEE Symp. Logic in Comp. Sci., pp. 399–410. IEEE Computer Society Press, Los Alamitos (2006)
19. Thomas, W.: Automata on infinite objects. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, ch. 4, pp. 135–191. Elsevier Science Publishers,North-Holland, Amsterdam (1990)

Linear Time Logics Around PSL: Complexity, Expressiveness, and a Little Bit of Succinctness

Martin Lange

Department of Computer Science, University of Aarhus, Denmark

Abstract. We consider linear time temporal logic enriched with semi-extended regular expressions through various operators that have been proposed in the literature, in particular in Accelera's Property Specification Language. We obtain results about the expressive power of fragments of this logic when restricted to certain operators only: basically, all operators alone suffice for expressive completeness w.r.t. ω-regular expressions, just the closure operator is too weak. We also obtain complexity results. Again, almost all operators alone suffice for EXPSPACE-completeness, just the closure operator needs some help.

1 Introduction

Pnueli has acquired the linear time temporal logic LTL from philosophy for the specification of runs of reactive systems [7]. It has since been established as a milestone in computer science, particularly in automatic program verification through model checking. Its success and popularity as such is not matched by its expressive power though which was shown to be rather limited: LTL is equi-expressive to First-Order Logic on infinite words and, thus, can express exactly the star-free ω-languages [5,11]. This is not enough for compositional verification for instance.

Some early attempts have been made at extending LTL with the aim of capturing all ω-regular languages. Wolper's ETL incorporates non-deterministic Büchi automata as connectives in the language [14]. QPTL extends LTL with quantifications over atomic propositions in the style of Monadic Second-Order Logic [10]. The linear time μ-calculus μTL allows arbitrary recursive definitions of properties via fixpoint quantifiers in the logic [2,13].

None of these logics has seen an impact that would make it replace LTL as the most popular specification language for linear time properties. This may be because of non-elementary complexity (QPTL), syntactical inconvenience (ETL), and difficulty in understanding specifications (μTL), etc.

Nevertheless, the need for a convenient temporal specification formalism for ω-regular properties is widely recognised. This is for example reflected in the definition of Accelera's *Property Specification Language* (PSL), designed to provide a general-purpose interface to hardware verification problems [1]. At its temporal layer it contains a logic that is capable of expressing all ω-regular properties. This is mainly achieved through the introduction of operators in LTL that take semi-extended regular expressions as arguments. Following common agreement in the temporal logic community we also use PSL to refer to this logic. To be more precise: PSL is often used to describe a temporal logic enhanced with semi-extended regular expressions through various operators. This is because the original definition of Accelera's PSL allows a multitude of operators of which some are just syntactical sugar. For example, the intersection operator on regular expressions that does not require the same length of its operands can easily be expressed using the usual intersection operator: $\alpha\&\beta \equiv (\alpha; \Sigma^*)\&\&(\beta; \Sigma^*)$, etc.

Much effort has already been taken to design verification algorithms for PSL properties, for example through automata-theoretic constructions [3]. Nevertheless, we are not aware of a rigorous analysis of its computational complexity, and the expressive power of fragments obtained by restricting the use of temporal operators in the logic. In order to make up for this, we consider linear time temporal logic enriched with various operators that have occurred in the literature so far, not only in PSL. For pragmatics and complexity-theoretic upper bounds it is clearly desirable to consider the richest possible logic – not just in terms of expressive power but mainly regarding its variety of temporal operators. We include the commonly known *until* operator from LTL and fixpoint quantifiers from μTL. We also allow a stengthened *until* operator $\mathtt{U}^\alpha$ from *Dynamic LTL* which – roughly speaking – asserts that the until must be satisfied at the end of a word in $L(\alpha)$ [6]. We include an *and-then* operator $\diamond\!\!\rightarrow$ which realises concatenation of a semi-extended regular expression with a temporal formula. It occurs in PSL in a slightly different way where the regular expression and the temporal formula are required to overlap in one state. This can, however, easily be defined using an $\mathtt{U}^\alpha$, and in order to cover as many cases as possible we let it denote the non-overlapping version here. Finally, inspired by the other PSL operator that links regular expressions with formulas we also include a *closure* operator $\mathtt{Cl}(\alpha)$. It asserts that at no position does a word look like it is not going to belong to $L(\alpha)$. In other words, all of its prefixes must be extendable to a finite word in $L(\alpha)$. This is slightly more general than the original PSL closure operator, see below for details.

Clearly, not all operators are needed in order to achieve expressive completeness w.r.t. ω-regular properties. For example, it is known that the specialised strengthened *until* operator $\mathtt{F}^\alpha$ is sufficient [6]. Sect. 3 shows what happens w.r.t. expressive power when certain operators are excluded from the logic. Because of lack of space here we only scrape upon the issue of succinctness. Clearly, since semi-extended regular expressions are only as expressive as ordinary ones there is no gain in expressive power from using them instead. However, some properties may be definable using shorter formulas. Note that Dynamic LTL for example, known to be PSPACE-complete [6], allows ordinary regular expressions only. Sect. 4 uses the expressiveness results to derive a general upper bound of det. exponential space for the satisfiability problem of this logic via a reduction to μTL. We also show that this is optimal obtaining the rule of thumb: regular expressions leave temporal logic PSPACE-complete, semi-extended ones make it EXPSPACE-complete.

2 Preliminaries

We start by recalling *semi-extended regular expressions*. Let $\mathcal{P} = \{q, p, \ldots\}$ be finite set of atomic propositions. *Propositional Logic* (PL) is the least set containing $\mathcal{P}$, the constants $\top, \bot$, and being closed under the operators $\vee, \wedge, \rightarrow$, etc., as well as the complementation operator $\bar{\cdot}$.

The satisfaction relation between symbols of the alphabet $2^{\mathcal{P}}$ and formulas of PL is the usual one: $a \models q$ iff $q \in a$; $a \models b_1 \wedge b_2$ iff $a \models b_1$ and $a \models b_2$; $a \models \overline{b}$ iff $a \not\models b$; etc.

We fix $\Sigma := 2^{\mathcal{P}}$ as an alphabet for the remainder of the paper. For a word $w \in \Sigma^*$ we use $|w|$ to denote its length, ϵ to denote the empty word, and w^i to denote the i-th symbol of w for $i \in \{0, \ldots, |w|-1\}$. The set of all infinite words is Σ^ω. For a finite or infinite word w we write $w^{i..j}$ to denote the subword $w^i \ldots w^{j-1}$ of length $j - i$, and $w^{i..}$ to denote its suffix starting with w^i. We write $w \prec v$ to denote that w is a proper prefix of v.

For two languages L, L' with $L \subseteq \Sigma^*$ and L' either a language of finite or of infinite words, their composition LL' denotes the concatenation of all words in L and L' in the usual way. The n-fold iteration (in the finite case) is denoted L^n with $L^0 = \{\epsilon\}$. An important mapping from languages of finite words to those of infinite ones is the *closure*, defined for all $L \subseteq \Sigma^*$ as $Cl(L) := \{w \in \Sigma^\omega \mid \forall k \in \mathbb{N}\ \exists v \in \Sigma^* \text{ s.t. } w^{0..k}v \in L\}$.

Semi-extended regular expressions (SERE) are now built upon formulas of PL in the following way.

$$\alpha \ ::= \ b \mid \alpha \cup \alpha \mid \alpha \cap \alpha \mid \alpha ; \alpha \mid \alpha^*$$

where $b \in$ PL. Let $|\alpha| := |Sub(\alpha)|$ measure the size of an SERE with $Sub(\alpha)$ being the set of all subexpressions occurring in α where propositional formulas are atomic. Another important measure is the number of intersections occurring in α: $is(\alpha) := |\{\beta \in Sub(\alpha) \mid \beta \text{ is of the form } \beta_1 \cap \beta_2\}$.

SERE define languages of finite words in the usual way. $L(b) := \{w \in \Sigma^* \mid |w| = 1, w_0 \models b\}$, $L(\alpha_1 \cup \alpha_2) := L(\alpha_1) \cup L(\alpha_2)$, $L(\alpha_1 \cap \alpha_2) := L(\alpha_1) \cap L(\alpha_2)$, $L(\alpha_1; \alpha_2) := L(\alpha_1)L(\alpha_2)$, and $L(\alpha^*) := \bigcup_{n \in \mathbb{N}} (L(\alpha))^n$.

The intersection operator $\cap$ allows us to define some usual ingredients of the syntax of regular expressions as abbreviations: $\emptyset := (q) \cap (\overline{q})$ for some $q \in \mathcal{P}$, and $\epsilon := (q)^* \cap (\overline{q})^*$. Equally, the symbolic encoding of symbols in PL enables a constant representation of the universal regular expression: $\Sigma^* = (\top)^*$.

An ordinary *regular expressions* (RE) is an SERE α with $is(\alpha) = 0$. We allow ϵ and $\emptyset$ as primitives in RE though. An *ω-regular expression* (ω-RE) γ is of the form $\bigcup_{i=1}^{n} \alpha_i; \beta_i^\omega$ for some $n \in \mathbb{N}$ s.t. α_i, β_i are RE, and $\epsilon \notin L(\beta_i)$. Its language $L(\gamma)$ is defined in the usual way using union, concatenation and infinite iteration of finite languages.

Next we consider linear time temporal logic. Take $\mathcal{P}$ from above. Let $\mathcal{V} = \{X, Y, \ldots\}$ be a countably infinite set of monadic second-order variables. Formulas of full *Linear Time Temporal Logic with SERE* ($\mathrm{TL}^{\mathrm{SERE}}$) are given by the following grammar.

$$\varphi \ ::= \ q \mid \varphi \wedge \varphi \mid \neg\varphi \mid X \mid \bigcirc\varphi \mid \varphi \, \mathtt{U} \, \varphi \mid \varphi \, \mathtt{U}^\alpha \, \varphi \mid \mu X.\varphi \mid \alpha \diamond\!\!\rightarrow \varphi \mid \mathtt{Cl}(\alpha)$$

where $q \in \mathcal{P}$, $X \in \mathcal{V}$, and α is an SERE over $\mathcal{P}$. As usual, we require variables X to only occur under the scope of an even number of negation symbols within ψ if X is quantified by $\mu X.\psi$. If the α are restricted to ordinary regular expressions we obtain the logic TL$^{\text{RE}}$. We also use the usual abbreviated Boolean operators like $\vee$, and the LTL-operators $\mathtt{F}$ and $\mathtt{G}$ as well as the strengthened version $\mathtt{F}^{\alpha}\,\varphi := \mathtt{tt}\,\mathtt{U}^{\alpha}\,\varphi$.

The size of a formula is $|\varphi| := |Sub(\varphi)|$ where $Sub(\varphi)$ is the set of all subformulas of φ including $Sub(\alpha)$ for any SERE α occurring in φ. The reason for not counting them as atomic is the expressive power of the temporal operators, c.f. next section below. It is possible to encode a lot of temporal property in the SERE rather than the temporal operators themselves. Counting SERE as atomic would therefore lead to unnaturally small formulas. The measure $is(\alpha)$ can then be extended to formulas straight-forwardly: $is(\varphi) := |\{\beta \in Sub(\varphi) \mid \beta$ is an SERE of the form $\beta_1 \cap \beta_2\}$.

TL$^{\text{SERE}}$ is interpreted over infinite words $w \in \Sigma^{\omega}$. The semantics assigns to each formula φ a language $L_\rho(\varphi) \subseteq \Sigma^{\omega}$ relative to some environment $\rho : \mathcal{V} \to 2^{\Sigma^{\omega}}$ which interprets free variables by languages of infinite words. We write $\rho[X \mapsto L]$ for the function that maps X to L and agrees with ρ on all other arguments.

$$\begin{array}{rcl}
L_\rho(q) & := & \{w \in \Sigma^{\omega} \mid q \in w^0\} \\
L_\rho(\varphi \wedge \psi) & := & L_\rho(\varphi) \cap L_\rho(\psi) \\
L_\rho(\neg\varphi) & := & \Sigma^{\omega} \setminus L_\rho(\varphi) \\
L_\rho(X) & := & \rho(X) \\
L_\rho(\bigcirc\varphi) & := & \{w \in \Sigma^{\omega} \mid w^{1..} \in L_\rho(\varphi)\} \\
L_\rho(\varphi\,\mathtt{U}\,\psi) & := & \{w \in \Sigma^{\omega} \mid \exists k \in \mathbb{N} \text{ s.t. } w^{k..} \in L_\rho(\psi) \text{ and } \forall j < k : \ w^{j..} \in L_\rho(\varphi)\} \\
L_\rho(\varphi\,\mathtt{U}^{\alpha}\,\psi) & := & \{w \in \Sigma^{\omega} \mid \exists k \in \mathbb{N} \text{ s.t. } w^{0..k+1} \in L(\alpha) \text{ and } w^{k..} \in L_\rho(\psi) \\
 & & \qquad\qquad \text{and } \forall j < k : \ w^{j..} \in L_\rho(\varphi)\} \\
L_\rho(\mu X.\varphi) & := & \bigcap\{L \subseteq \Sigma^{\omega} \mid L_{\rho[X \mapsto L]}(\varphi) \subseteq L\} \\
L_\rho(\alpha \diamond\!\!\rightarrow \varphi) & := & L(\alpha)L_\rho(\varphi) \\
L_\rho(\mathtt{Cl}(\alpha)) & := & Cl(L(\alpha))
\end{array}$$

The formula $\mu X.\psi$ defines the least fixpoint of the monotone function which maps a language L of infinite words to the set of words that satisfy ψ under the assumption that X defines L. Due to negation closure and fixpoint duality we can also define greatest fixpoints of such maps via $\nu X.\psi := \neg\mu X.\neg\psi[\neg X/X]$, where the latter denotes the simultaneous substitution of $\neg X$ for every occurrence of X in ψ.

As usual, we write $\varphi \equiv \psi$ if for all environments ρ we have $L_\rho(\varphi) = L_\rho(\psi)$. We use the notation TL[..], listing certain operators, to denote syntactical fragments of TL$^{\text{SERE}}$, for instance TL$^{\text{SERE}}$ $=$ TL$[\bigcirc, \mathtt{U}, \mathtt{U}^{\alpha}, \mu, \diamond\!\!\rightarrow, \mathtt{Cl}]$. We do not list variables explicitly because variables without quantifiers or vice-versa are meaningless. Also, we always assume that these fragments contain atomic propositions and the

Boolean operators. For example, LTL is TL$[\bigcirc, \mathtt{U}]$, and it has genuine fragments TL$[\bigcirc, \mathtt{F}]$, TL$[\mathtt{F}]$, TL$[\bigcirc]$ [4]. The linear time μ-calculus μTL is TL$[\bigcirc, \mu]$, and PSL is TL$[\mathtt{U}, \mathtt{F}^\alpha, \mathtt{Cl}]$: the PSL formula consisting of an SERE α is equivalent to the TL$^{\text{SERE}}$ formula $(\mathtt{F}^\alpha\, \mathtt{tt}) \vee \mathtt{Cl}(\alpha)$, where $\mathtt{tt} := q \vee \neg q$ for some $q \in \mathcal{P}$. If we restrict the use of semi-extended regular expressions in a fragment to ordinary regular expressions only then we denote this by TL$^{\text{RE}}[..]$.

Some of the results regarding expressive power are obtained using automata on finite words, Büchi automata, and translations from SERE into those. We therefore quickly recall the automata-theoretic foundations. A *nondeterministic finite automaton* (NFA) is a tuple $\mathcal{A} = (Q, \mathcal{P}, q_0, \delta, F)$ with Q a finite set of states, $q_0 \in Q$ a starting state, and $F \subseteq Q$ a set of final states. Its alphabet is $\Sigma = 2^{\mathcal{P}}$, and its transition relation δ is a finite subset of $Q \times \text{PL} \times Q$. We define $|\mathcal{A}| := |\delta|$ as the size of the NFA. Note that these NFA use symbolic representations of symbols in their transitions in order to avoid unnecessary exponential blow-ups in their sizes. A run of $\mathcal{A}$ on a finite word $w \in \Sigma^*$ is a non-empty sequence $q_0, w^0, q_1, w^1, \ldots, q_n$ s.t. for all $i = 0, \ldots, n-1$ there is a b s.t. $w^i \models b$ and $(q_i, b, q_{i+1}) \in \delta$. It is accepting iff $q_n \in F$.

Translating RE into NFA is a standard exercise. It is not hard to see that the usual uniform translation applies to the case of symbolically represented transition relations as well. It is also relatively easy to see that it can be extended to SERE. The translation of the SERE $\alpha \cap \beta$ simply uses the well-known product construction on NFA. Clearly, the size of the product NFA is quadratic in the original sizes, and iterating this leads to an exponential translation.

Proposition 1. *For every SERE α there is an NFA $\mathcal{A}_\alpha$ s.t. $L(\mathcal{A}_\alpha) = L(\alpha)$ and $|\mathcal{A}_\alpha| = O(2^{|\alpha| \cdot (is(\alpha)+1)})$.*

To see that an exponential blow-up can occur simply consider the recursively defined RE α_n with $\alpha_0 := q$ and $\alpha_{n+1} := \alpha_n; \neg q; \alpha_n; q$.

A *nondeterministic Büchi automaton* (NBA) is syntactically defined like an NFA. It runs on infinite words $w \in \Sigma^\omega$ via $q_0, w^0, q_1, w^1, q_2, \ldots$ s.t. for all $i \in \mathbb{N}$ there is a $b \in \text{PL}$ s.t. $w^i \models b$ and $(q_i, b, q_{i+1}) \in \delta$. It is accepting iff there are infinitely many i s.t. $q_i \in F$. We use NBA to model the closure of languages L of finite words. This is particularly easy if L is given by an NFA.

Lemma 1. *For every NFA $\mathcal{A}$ there is an NBA $\mathcal{A}_{cl}$ s.t. $L(\mathcal{A}_{cl}) = Cl(L(\mathcal{A}))$, and $|\mathcal{A}_{cl}| \leq |\mathcal{A}|$.*

Proof. Let $\mathcal{A} = (Q, \mathcal{P}, q_0, \delta, F)$ and $Q' \subseteq Q$ consist of all states which are reachable from q_0 and productive, i.e. from each of them a final state is reachable. Then define $\mathcal{A}_{cl} := (Q', \mathcal{P}, q_0, \delta, Q')$. We have $L(\mathcal{A}_{cl}) \subseteq Cl(L(\mathcal{A}))$ because runs in $\mathcal{A}_{cl}$ only visit states from with final states in $\mathcal{A}$ are reachable.

For the converse direction suppose $w \in Cl(L(\mathcal{A}))$, i.e. for every $k \in \mathbb{N}$ there is a $v \in \Sigma^*$ and an accepting run of $\mathcal{A}$ on $w^{0..k}v$. Clearly, all these runs stay in Q'. Furthermore, they can be arranged to form an infinite but finitely branching tree, and König's Lemma yields an accepting run of $\mathcal{A}_{cl}$ on w. $\square$

3 Expressive Power and Succinctness

For two fragments $\mathcal{L}_1$ and $\mathcal{L}_2$ of $\mathrm{TL}^{\mathrm{SERE}}$ we write $\mathcal{L}_1 \leq_{f(n,k)} \mathcal{L}_2$ if for every $\varphi \in \mathcal{L}_1$ there is a $\psi \in \mathcal{L}_2$ s.t. $\varphi \equiv \psi$ and $|\psi| \leq f(|\varphi|, \mathit{is}(\varphi))$. This measures relative expressive power and possible succinctness. We write $\mathcal{L}_1 \equiv_{f(n,k)} \mathcal{L}_2$ if $\mathcal{L}_1 \leq_{f(n,k)} \mathcal{L}_2$ and $\mathcal{L}_2 \leq_{f(n,k)} \mathcal{L}_1$. In case the succinctness issue is unimportant we simply denote expressive subsumption and equivalence using $\leq$ and $\equiv$.

It is well-known that $\mathrm{TL}[\bigcirc, \mu]$, the linear time μ-calculus, has the same expressive power as Monadic Second-Order Logic on infinite words, or equivalently, ω-RE. It is also known that $\mathrm{TL}^{\mathrm{RE}}[\mathtt{F}^\alpha]$ and, thus, $\mathrm{TL}[\mathtt{F}^\alpha]$ is of the same expressiveness [6]. We start by showing that the non-overlapping *and-then* operator suffices for expressive completeness too.

Theorem 1. $\mathrm{TL}[\mathtt{F}^\alpha] \leq_{O(n^{k+1})} \mathrm{TL}[\diamond\!\!\rightarrow]$.

Proof. We use a function $\mathit{cut} : \mathrm{SERE} \rightarrow 2^{\mathrm{SERE}\times\mathrm{PL}}$ that decomposes an SERE via $\mathit{cut}(\alpha) = \{(\beta_1, b_1), \ldots, (\beta_n, b_n)\}$ only if $L(\alpha) = \bigcup_{i=1}^{n} L(\beta_i)L(b_i)$. This can be recursively defined as

$$\begin{array}{rcl}
\mathit{cut}(b) & := & \{(\epsilon, b)\} \\
\mathit{cut}(\alpha_1 \cup \alpha_2) & := & \mathit{cut}(\alpha_1) \cup \mathit{cut}(\alpha_2) \\
\mathit{cut}(\alpha_1 \cap \alpha_2) & := & \{(\beta_1 \cap \beta_2, b_1 \wedge b_2) \mid (\beta_i, b_i) \in \mathit{cut}(\alpha_i) \text{ for } i = 1, 2\} \\
\mathit{cut}(\alpha_1; \alpha_2) & := & \{(\alpha_1; \beta, b) \mid (\beta, b) \in \mathit{cut}(\alpha_2)\} \cup \begin{cases} \mathit{cut}(\alpha_1) & , \text{ if } \epsilon \in L(\alpha_2) \\ \emptyset & , \text{ o.w.} \end{cases} \\
\mathit{cut}(\alpha^*) & := & \{(\alpha^*; \beta, b) \mid (\beta, b) \in \mathit{cut}(\alpha)\}
\end{array}$$

Note that $|\mathit{cut}(\alpha)| \leq O(|\alpha|^{\mathit{is}(\alpha)+1})$. We can now use this decomposition to translate a formula of the form $\mathtt{F}^\alpha\, \varphi$ with an overlap between α and φ into a non-overlapping one: $\mathtt{F}^\alpha\, \varphi \equiv \bigvee_{(\beta,b)\in \mathit{cut}(\alpha)} \beta \diamond\!\!\rightarrow (b \wedge \varphi)$. □

The same reduction can be carried out from $\mathrm{TL}^{\mathrm{RE}}[\mathtt{F}^\alpha]$ to $\mathrm{TL}^{\mathrm{RE}}[\diamond\!\!\rightarrow]$, and it would be polynomial because we would have $k = 0$.

The expressive power of the closure operator is a natural concern. It turns out to be weaker than $\mathtt{U}^\alpha$ or $\diamond\!\!\rightarrow$ for instance. We use an invariant technique on automata structure to show this. An NBA $\mathcal{A} = (Q, \mathcal{P}, q_0, \delta, F)$ is called $\exists\forall$-accepting if there is no $q \in F$ from which a $q' \notin F$ is reachable. Runs of these automata accept iff they eventually only visit final states. Note that this is not the same as a co-Büchi automaton since it is a syntactical restriction, but it is a special case of a *weak* NBA. An important observation is that the NBA $\mathcal{A}_{cl}$ as constructed in the proof of Lemma 1 are actually $\exists\forall$-accepting.

Lemma 2. *Every language definable in* $\mathrm{TL}[\mathtt{Cl}]$ *can be recognised by an* $\exists\forall$-*accepting NBA.*

Proof. Let $\psi \in \mathrm{TL}[\mathtt{Cl}]$. Clearly, ψ is a Boolean combination of formulas of the form $\mathtt{Cl}(\alpha)$. Using deMorgan laws it is possible to transform ψ into a positive Boolean combination of formulas of the form $\mathtt{Cl}(\alpha)$ and $\neg\mathtt{Cl}(\beta)$.

Now note that $w \not\models \mathtt{Cl}(\beta)$ iff there is a $v \prec w$ s.t. for all $v' \in \Sigma^*$: $vv' \notin L(\beta)$. Let $\mathcal{A}_\beta$ be the NFA that recognises exactly the models of β, c.f. Prop. 1. It can be determinised and complemented into a DFA $\mathcal{A}_{\overline{\beta}} = (Q, \mathcal{P}, q_0, \delta, F)$. Let $Q' := \{q \in Q \mid \forall q' \in Q$: if q' is reachable from q then $q' \in F\}$. Note that $Q' \subseteq F$. Now consider the deterministic NBA $\mathcal{A}'_{\overline{\beta}} = (Q, \mathcal{P}, q_0, \delta, Q')$. It accepts a word w iff the unique run on it exhibits a prefix v which takes the NBA from q_0 to a state $q \in F$. Hence, $v \notin L(\beta)$. Furthermore, $vv' \notin L(\beta)$ for any v' because $\mathcal{A}'_{\overline{\beta}}$ is deterministic and only states in F are reachable from q.

This shows that ψ is equivalent to a positive Boolean combination of languages recognisable by $\exists\forall$-accepting NBA. Given two $\exists\forall$-accepting NBA recognising L_1 and L_2 it is easy to construct $\exists\forall$-accepting NBA for the languages $L_1 \cup L_2$ and $L_1 \cap L_2$ using the standard union and product constructions. Hence, $L(\psi)$ can be recognised by an $\exists\forall$-accepting NBA. $\square$

Lemma 3. $\mathrm{TL}[\bigcirc, \mathtt{Cl}] \equiv_{O(n)} \mathrm{TL}[\mathtt{Cl}]$.

Proof. The $\geq$ part is of course trivial. For the $\leq$ part first note that because of the equivalences $\bigcirc\neg\psi \equiv \neg\bigcirc\psi$ and $\bigcirc(\psi_1 \wedge \psi_2) \equiv \bigcirc\psi_1 \wedge \bigcirc\psi_2$ every $\mathrm{TL}[\bigcirc, \mathtt{Cl}]$ formula φ can be transformed into a Boolean combination of atomic formulas of the form $\bigcirc^k q$ or $\bigcirc^k \mathtt{Cl}(\alpha)$ for some SERE α and some $k \in \mathbb{N}$. Using the equivalences $q \equiv \mathtt{Cl}(q; \top^*)$ and $\bigcirc\mathtt{Cl}(\alpha) \equiv \mathtt{Cl}(\top; \alpha)$ it is then possible to eliminate all occurrences of the $\bigcirc$ operators. The resulting formula is clearly of linear size. $\square$

The following looks like an immediate consequence of this but it needs the observation that the $\bigcirc$ operator commutes with an $\mathtt{U}$. The same holds for the $\mathtt{F}$ operator.

Lemma 4. $\mathrm{TL}[\bigcirc, \mathtt{U}, \mathtt{Cl}] \equiv_{O(n)} \mathrm{TL}[\mathtt{U}, \mathtt{Cl}]$.

Proof. As the proof of Lemma 3 but also using the equivalence $\bigcirc(\varphi \,\mathtt{U}\, \psi) \equiv (\bigcirc\varphi)\,\mathtt{U}\,(\bigcirc\psi)$ in order to push the $\bigcirc$ operators down to atomic propositions and closure operators where they can be eliminated. $\square$

Corollary 1. $\mathrm{TL}[\bigcirc, \mathtt{F}, \mathtt{Cl}] \equiv_{O(n)} \mathrm{TL}[\mathtt{F}, \mathtt{Cl}]$.

Theorem 2. $\mathrm{TL}[\mathtt{Cl}]$ *is $\leq$-incomparable to both* $\mathrm{TL}[\bigcirc, \mathtt{U}]$ *and* $\mathrm{TL}[\mathtt{F}]$.

Proof. Since $\mathrm{TL}[\mathtt{F}] \leq \mathrm{TL}[\bigcirc, \mathtt{U}]$ it suffices to show that there is a $\mathrm{TL}[\mathtt{Cl}]$ property which is not definable in $\mathrm{TL}[\bigcirc, \mathtt{U}]$, and a $\mathrm{TL}[\mathtt{F}]$ property which is not definable in $\mathrm{TL}[\mathtt{Cl}]$.

It is well-known that $\mathrm{TL}[\bigcirc, \mathtt{U}]$, aka LTL, cannot express "q holds in every even moment". This, however, can easily be expressed by $\mathtt{Cl}((q; \top)^*)$. For the converse direction take the $\mathrm{TL}[\mathtt{F}]$ formula $\mathtt{G}\,\mathtt{F}\,q$. According to Lemma 2 its language would be recognisable by an $\exists\forall$-accepting NBA $\mathcal{A} = (Q, \mathcal{P}, q_0, \delta, F)$ if it was definable in $\mathrm{TL}[\mathtt{Cl}]$. Let $n := |Q|$. Consider the word $w := (\{q\}\emptyset^{n+1})^\omega$. Since $w \in L(\mathtt{G}\,\mathtt{F}\,q)$ there is an accepting run $q_0, w^0, q_1, w^1, \ldots$ of $\mathcal{A}$ on w. Since $\mathcal{A}$ is $\exists\forall$-accepting there is a k s.t. $q_i \in F$ for all $i \geq k$. Then there will be $j > i \geq k$ s.t. $q_i = q_j$ and $w^h = \emptyset$ for all $h = i, \ldots, j-1$. This gives us an accepting run on a word of the form $(\{q\}\emptyset)^*\emptyset^\omega$ which should not be accepted. $\square$

Corollary 2. TL[$\mathtt{F}, \mathtt{Cl}$] $\not\leq$ TL[$\bigcirc, \mathtt{U}$].

The converse direction is still open. It remains to be seen whether or not every LTL-definable property can also be expressed in TL[$\mathtt{F}, \mathtt{Cl}$]. Note for example that, for atomic propositions p and q, we have $p \, \mathtt{U} \, q \equiv \mathtt{F} \, q \wedge \mathtt{Cl}(p^*; q; \top^*)$. However, this does not extend easily to arbitrary formulas of the form $\varphi \, \mathtt{U} \, \psi$.

Clearly, adding the operator $\diamond\!\!\rightarrow$ or $\mathtt{F}^\alpha$ to the closure operator yields expressive completeness since these alone are sufficient for that. This poses the question of what happens when the closure operator is combined with LTL operators which are not enough to achieve ω-regular expressiveness. One answer is obtained easily from Thm. 2: TL[$\bigcirc, \mathtt{U}$] is strictly less expressive than TL[$\mathtt{U}, \mathtt{Cl}$]. On the other hand, we suspect that TL[$\mathtt{U}, \mathtt{Cl}$] is also strictly contained in TL$^{\text{SERE}}$. In particular, we believe that the property "q holds infinitely often in even moments" is not expressible in TL[$\mathtt{U}, \mathtt{Cl}$].

It is known that each operator present in TL$^{\text{SERE}}$ does not exceed its expressive power beyond ω-regularity. Therefore it is fair to assume that TL$^{\text{SERE}}$ is only as expressive as μTL. We present a direct and conceptionally simple translation from TL$^{\text{SERE}}$ to TL[$\bigcirc, \mu$] which forms the basis for the complexity analysis in the next section. The following lemmas prepare for the not so straight-forward cases in that translation.

Lemma 5. *For every NBA $\mathcal{A}$ there is a closed* TL[$\bigcirc, \mu$] *formula $\varphi_{\mathcal{A}}$ s.t. $L(\varphi_{\mathcal{A}}) = L(\mathcal{A})$ and $|\varphi_{\mathcal{A}}| = O(|\mathcal{A}|)$.*

Proof. Let $\mathcal{A} = (Q, \mathcal{P}, q_0, \delta, F)$. W.l.o.g. we assume $Q = \{0, \ldots, n\}$ for some $n \in \mathbb{N}$, $q_0 = 0$, and $F = \{0, \ldots, m\}$ for some $m \leq n$. Note that the starting state can always be assumed to be final by adding a copy of the starting state which is not reachable from any other state.

The construction of $\varphi_{\mathcal{A}}$ uses n monadic second-order variables X_i, $i \in Q$, each X_i representing the moments of a run in which $\mathcal{A}$ is in state i. To understand the construction best we introduce an auxiliary device of an NBA $\mathcal{A}_\rho$ with a partial function $\rho : Q \to 2^{\Sigma^\omega}$ acting as an oracle.[1] $\mathcal{A}_\rho$ immediately accepts the language $\rho(i)$ upon entrance of state i when $\rho(i)$ is defined.

For every $i \in Q$ in the order $i = n, \ldots, 0$ we construct a formula $\psi_i(X_0, \ldots, X_{i-1})$ s.t. $L_\rho(\psi_i) = \{w \mid \mathcal{A}_{\rho'}$ accepts w starting in state $i\}$ where $\rho'(j) := \rho(X_j)$ for all $j < i$.

$$\psi_i \quad := \quad \sigma_i X_i . \bigvee_{(i,b,j) \in \delta} b \wedge \bigcirc \begin{cases} \psi_j & , \text{ if } j > i \\ X_j & , \text{ o.w.} \end{cases}$$

where $\sigma_i := \mu$ if $i > m$ and ν otherwise. The correctness claim above is straight-forwardly verified by induction on i. Also note that ψ_i is well-defined, and ψ_0 is a closed formula. Its correctness claim refers to the oracle NBA $\mathcal{A}_{\rho'}$ starting in the initial state 0 and not using the oracle at all. Hence, we have $L_\rho(\psi_0) = L(\mathcal{A})$ for any ρ, and therefore define $\varphi_{\mathcal{A}} := \psi_0$. Note that its size is linear in the size of $\mathcal{A}$. □

[1] Since oracles are for automata what environments are for the semantics function – an interpreter of open parts – we use the same symbol ρ here.

Lemma 6. *For every NFA $\mathcal{A}$ and every* $\mathrm{TL}[\bigcirc,\mu]$ *formula* χ *there is a* $\mathrm{TL}[\bigcirc,\mu]$ *formula* $\varphi_{\mathcal{A},\chi}$ *s.t. for any environment* ρ*:* $L_\rho(\varphi_{\mathcal{A},\chi}) = L(\mathcal{A})L_\rho(\chi)$ *and* $|\varphi_{\mathcal{A},\chi}| = O(|\mathcal{A}| + |\chi|)$*.*

Proof. Similar to the previous construction. Let $\mathcal{A} = (Q, \mathcal{P}, q_0, \delta, F)$ with $Q = \{0, \ldots, n\}$. We construct for every $i = n, \ldots, 0$ a $\mathrm{TL}[\bigcirc,\mu]$ formula $\psi_i(X_0, \ldots, X_{i-1})$ s.t. $L_\rho(\psi_i) = L_i L_\rho(\chi)$ where $L_i \subseteq \Sigma^*$ consists of all words that are accepted by $\mathcal{A}$ when starting in state i under the assumption that upon entering state $j < i$, it immediately accepts the language $\rho(X_j)$.

$$\psi_i \;:=\; \mu X_i. \bigvee_{(i,b,j)\in\delta} \chi_i \vee (b \wedge \bigcirc \begin{cases} \psi_j & \text{, if } j > i \\ X_j & \text{, o.w.} \end{cases})$$

where $\chi_i := \chi$ if $i \in F$, and $\mathtt{ff}$ otherwise. Note that the language defined by an NFA is given as the simultaneous least fixpoint of the languages recognised by each state. Hence, only μ quantifiers are needed here as opposed to the NBA case above where the language is given as a nested greatest/least fixpoint.

Again, define $\varphi_{\mathcal{A},\chi}$ as ψ_0. The correctness of this construction w.r.t. to the specification above can straight-forwardly be proved by induction on i. Also, the claim on its size is easily checked to be true. $\square$

Lemma 7. *For every* $\mathrm{TL}[\bigcirc,\mu]$ *formulas* φ_1, φ_2 *and every NFA* $\mathcal{A}$ *there is a* $\mathrm{TL}[\bigcirc,\mu]$ *formula* $\varphi_{\mathcal{A},\varphi_1,\varphi_2}$ *s.t.* $|\varphi_{\mathcal{A},\varphi_1,\varphi_2}| = O(|\mathcal{A} + |\varphi_1| + |\varphi_2|)$ *and for all environments* ρ*:* $L_\rho(\varphi_{\mathcal{A},\varphi_1,\varphi_2}) = \{w \in \Sigma^\omega \mid \exists k \in \mathbb{N}$ *s.t.* $w^{0..k+1} \in L(\mathcal{A})$ *and* $w^{k..} \in L_\rho(\varphi_2)$ *and* $\forall j < k : w^{j..} \in L_\rho(\varphi_1)\}$*.*

Proof. This is done in very much the same way as in the proof of Lemma 6. There are only two minor differences. (1) Because of the overlap between the part accepted by $\mathcal{A}$ and the part satisfying φ_2 we need to assert φ_2 not after having been in a final state but before entering one. (2) In every step that $\mathcal{A}$ does without entering a final state we need to require φ_1 to hold.

Again, let $\mathcal{A} = (Q, \mathcal{P}, q_0, \delta, F)$ with $Q = \{0, \ldots, n\}$. Define for each $i \in Q$:

$$\psi_i \;:=\; \mu X_i. \bigvee_{(i,b,j)\in\delta} b \wedge \big(\chi_i \vee (\varphi_1 \wedge \bigcirc \left\{ \begin{matrix} \psi_j & \text{, if } j > i \\ X_j & \text{, o.w.} \end{matrix} \right\})\big)$$

where $\chi_i := \varphi_2$ if $j \in F$, and $\chi_i := \mathtt{ff}$ otherwise. Note that the ability to prove φ_2 is linked to the fact whether or not j belongs to F because of the aforementioned overlap. Again, define $\varphi_{\mathcal{A},\varphi_1,\varphi_2} := \psi_0$ to finish the claim. $\square$

Theorem 3. $\mathrm{TL}^{\mathrm{SERE}} \leq_{O(2^{n\cdot(k+1)})} \mathrm{TL}[\bigcirc,\mu]$.

Proof. By induction on the structure of the formula. The claim is trivially true for atomic propositions and variables. Boolean operators, the temporal $\bigcirc$, and fixpoint quantifiers μ are translated uniformly. An $\mathtt{U}$ operator can be replaced by a least fixpoint formula. The remaining cases are the interesting ones.

Suppose $\varphi = \alpha \diamond\!\!\!\rightarrow \psi$. By hypothesis, there is a $\psi' \in \mathrm{TL}[\bigcirc,\mu]$ s.t. $\psi' \equiv \psi$ and $|\psi'| \leq O(2^{|\psi|\cdot(is(\psi)+1)})$. According to Prop. 1 there is an NFA $\mathcal{A}_\alpha$ s.t.

$L(\mathcal{A}_\alpha) = L(\alpha)$ and $|\mathcal{A}_\alpha| \leq O(2^{|\alpha|\cdot(is(\alpha)+1)})$. According to Lemma 6 there is a $\mathrm{TL}[\bigcirc,\mu]$ formula φ' s.t. $L_\rho(\varphi') = L(\mathcal{A}_\alpha)L_\rho(\psi')$ under any ρ. Thus, $L_\rho(\varphi') = L_\rho(\varphi)$. Furthermore, $|\varphi'| \leq O(2^{|\psi|\cdot(is(\psi)+1)} + 2^{|\alpha|\cdot(is(\alpha)+1)}) \leq O(2^{|\varphi|\cdot(is(\varphi)+1)})$.

The cases of $\varphi = \psi_1 \,\mathtt{U}^\alpha\, \psi_2$ and $\varphi = \mathtt{Cl}(\alpha)$ are done in the same way but using Lemmas 7, 1, and 5 instead. □

4 The Complexity of $\mathrm{TL}^{\text{SERE}}$ and Its Fragments

We can now easily obtain an upper bound on the complexity of the satisfiability problem for $\mathrm{TL}^{\text{SERE}}$ from Thm. 3. It composes the exponential reduction with the known PSPACE upper bound for μTL, found many times in different ways.

Proposition 2. [9,13] *Satisfiability in* $\mathrm{TL}[\bigcirc,\mu]$ *is PSPACE-complete.*

Corollary 3. *Satisfiability in (a)* $\mathrm{TL}^{\text{SERE}}$ *is in EXPSPACE, and (b)* TL^{RE} *is in PSPACE.*

Part (b) is not necessarily surprising. The satisfiability problem of all the logics considered in the literature with some of the operators of TL^{RE} can be decided in PSPACE, c.f. Prop. 2 and [9,6,3]. There is no need to assume that the combination of these operators should exceed the PSPACE bound.

Part (a) entails that PSL can be decided in deterministic exponential space. There is a translation of PSL into NBA of doubly exponential size – but measuring the size of regular expressions as their syntactical length – which goes via weak alternating Büchi automata [3]. It does not mention SERE, however, this can easily be incorporated, see above. Since the emptiness problem for NBA is known to be in NLOGSPACE, part (a) follows for PSL also from that translation and Savitch's Theorem [8].

A surprising fact is part (a) in conjunction with the emptiness problem for SERE. An immediate consequence of Prop. 1 is a PSPACE upper bound on the emptiness problem (i.e. satisfiability) of semi-extended regular expressions. However, combining that with the PSPACE decidable operators from temporal logic raises the space complexity by one exponential. This is also optimal. We will prove a lower complexity bound of deterministic exponential space by a reduction from the following 2^n-tiling problem, known to be EXPSPACE-hard [12]. Given an $n \in \mathbb{N}$ and a finite set $T = \{1,\ldots,m\}$ called tiles, and two binary relations $M_h, M_v \subseteq T \times T$ (for horizontal and vertical matching), decide whether or not there is a function $\tau : \{0,\ldots,2^n-1\} \times \mathbb{N} \to T$ s.t.

- $\forall i \in \{0,\ldots,2^n-2\}, \forall j \in \mathbb{N} : (\tau(i,j), \tau(i+1,j)) \in M_h$
- $\forall i \in \{0,\ldots,2^n-1\}, \forall j \in \mathbb{N} : (\tau(i,j), \tau(i,j+1)) \in M_v$

Note that such a function tiles the infinite corridor of width 2^n s.t. adjacent tiles match in the horizontal and vertical relations. In the following we will write iM_x for $\{j \mid (i,j) \in M_x\}$ where $x \in \{h,v\}$.

Theorem 4. *Satisfiability in* $\mathrm{TL}[\bigcirc,\mathtt{F},\mathtt{F}^\alpha]$ *is EXPSPACE-hard.*

Proof. By a reduction from the 2^n-tiling problem. Suppose n and a set T of tiles are given. We can easily regard $\{0,\dots,2^n-1\} \times \mathbb{N}$ as an infinite word in which the cell (i,j) is represented by the $(j \cdot 2^n + i)$-th symbol in that word. We will use atomic propositions $t_1,\dots,t_m$ to model the tiles and $c_0,\dots,c_{n-1}$ to enumerate the positions in the word modulo 2^n.

First of all we need a counter formula that axiomatises the enumeration of the symbols. It asserts that the cell at (i,j) is labeled with those propositions which represent set bits in the binary encoding of i. We use the fact that in binary increment a bit becomes set iff its current value correctly indicates whether the bit below does not decrease its value.

$$\varphi_{count} \;:=\; \chi_0 \;\wedge\; \mathtt{G}\Big(\,(c_0 \leftrightarrow \bigcirc\neg c_0) \;\wedge \bigwedge_{i=1}^{n-1} \bigcirc c_i \leftrightarrow \big(\; c_i \leftrightarrow (c_{i-1} \rightarrow \bigcirc c_{i-1})\;\big)\,\Big)$$

where $\chi_0 := \bigwedge_{i=0}^{n-1} \neg c_i$ marks the beginning of each row. We also need to say that each cell contains exactly one tile.

$$\varphi_{tile} \;:=\; \mathtt{G}\,(\bigvee_{i=1}^{m} t_i \wedge \bigwedge_{j \neq i} \neg t_j)$$

In order to compare vertically adjacent tiles we create an SERE α_n s.t. $L(\alpha_n) = \{w \mid |w| = 2^n + 1\}$. This is particularly easy on models which also satisfy φ_{count}. It suffices to require all counter bits to have the same value in the first and last symbol of each word, and to contain at most one symbol which satisfies χ_0 unless this is what it starts with.[2]

$$\alpha_n \;:=\; \top;(\neg\chi_0)^*;\chi_0;(\neg\chi_0)^* \;\cap\; \bigcap_{i=0}^{n-1} (c_i;\top^*;c_i) \cup (\neg c_i;\top^*;\neg c_i)$$

At last we need to axiomatise the two relations modelling the matching of tiles.

$$\varphi_h \;:=\; \mathtt{G}\,(\,\bigwedge_{i=1}^{m} t_i \;\rightarrow\; \bigcirc(\chi_0 \vee \bigvee_{j \in iM_h} t_j)\,)$$

$$\varphi_v \;:=\; \mathtt{G}\,(\,\bigwedge_{i=1}^{m} t_i \;\rightarrow\; \mathtt{F}^{\alpha_n}(\,\bigvee_{j \in iM_v} t_j)\,)$$

Then the given tiling problem has a positive instance iff $\varphi := \varphi_{count} \wedge \varphi_{tile} \wedge \varphi_h \wedge \varphi_v$ is satisfiable. Note that φ can be constructed in logarithmic space from the tiling problem and n. $\square$

This is not optimal in terms of the operators that are being used. They are just chosen to make the presentation easiest. Now note that the only temporal

[2] Given our definition of size of an SERE we could also recursively define $\alpha_n := \alpha_{n-1};\alpha_{n-1}$, etc. However, the definition we use here also shows EXPSPACE-hardness when the size is measured as the syntactical length.

operators occurring in that formula are $\bigcirc$, $\mathtt{G}$, and $\mathtt{F}^{\alpha}$. Hence, in order to reduce the number of operators used, and to strengthen the hardness result we simply need to express these operators in terms of others.

Corollary 4. *Satisfiability in* TL[$\mathtt{F}^{\alpha}$] *is EXPSPACE-hard.*

Proof. Because of $\bigcirc\varphi \equiv \mathtt{F}^{\top;\top}\,\varphi$ and $\mathtt{G}\,\varphi \equiv \neg(\mathtt{F}^{\top^*}\,\neg\varphi)$. □

Unfortunately, Thm. 1 does not immediately yield a lower bound for the fragment built upon the non-overlapping *and-then* operator as well. Remember that the translation from TL[$\mathtt{F}^{\alpha}$] to TL[$\diamond\!\!\rightarrow$] is exponential in the number of intersection operators. Nevertheless, the result can be restored.

Theorem 5. *Satisfiability in* TL[$\diamond\!\!\rightarrow$] *is EXPSPACE-hard.*

Proof. Consider the SERE

$$\begin{aligned}\alpha'_n \;:=\; & \big(\; c^*_{n-1}; (\neg c_{n-1})^*; c^*_{n-1} \cup (\neg c_{n-1})^*; c^*_{n-1}; (\neg c_{n-1})^* \;\big) \\ & \cap \Big(\; \chi_0; \top^*; (\bigwedge_{i=0}^{n} c_i) \cup \\ & \quad \bigcup_{i=0}^{n-1} c_i; \top^*; \neg c_i \cap (\bigcap_{j>i} (c_j; \top^*; c_j) \cup (\neg c_j; \top^*; \neg c_j)) \cap (\bigcap_{j<i} \neg c_j; \top^*; c_j) \;\Big)\end{aligned}$$

It asserts that there is a bit which is now 1 and 0 in the end, all higher bits have the same value now and then, and all lower bits change from 0 to 1. There is the special case of the counter being at value 0 now and at value $2^n - 1$ at the end. Also, we make sure that the SERE is only fulfilled by minimal words. Here this simply means that the highest bit changes its value at most twice. Then α'_n is fulfilled exactly by subwords of length 2^n in the context of φ_{count} from the proof of Thm. 4. Hence, we can prove this claim in exactly the same way but using $\alpha'_n \diamond\!\!\rightarrow \psi$ instead of $\mathtt{F}^{\alpha_n}\,\psi$. Furthermore, $\bigcirc\varphi \equiv \top \diamond\!\!\rightarrow \varphi$ and $\mathtt{G}\,\varphi \equiv \neg(\top^* \diamond\!\!\rightarrow \neg\varphi)$. □

One question arises naturally: does the closure operator alone suffice to gain EXPSPACE-hardness? The proof of the following theorem reformulates the reduction in the proof of Thm. 4 using the closure operator. However, the vertical matching relation in the tiling requires a single occurrence of a $\mathtt{G}$. Without this $\mathtt{G}$ operator we can only establish PSPACE-hardness. This does not follow from PSPACE-hardness of LTL, c.f. Thm. 2 above.

Theorem 6. *Satisfiability in* TL[$\mathtt{F}, \mathtt{Cl}$] *is EXPSPACE-hard.*

Proof. We simply replace the definitions of the four conjuncts in the constructed formula φ from the proof of Thm. 4. The first two are relatively easy to transform.

$$\varphi_{count} \;:=\; \chi_0 \wedge \mathtt{Cl}((\alpha'_n)^*) \qquad\qquad \varphi_{tile} \;:=\; \mathtt{Cl}(\;(\bigvee_{i=1}^{m} t_i \wedge \bigwedge_{j\neq i} \neg t_j)^*\;)$$

where α'_n is taken from the proof of Thm. 5 above.

Now consider the RE $\beta_h := \bigcup_{i=1}^{m} t_i; (\chi_0 \cup \bigcup_{j \in iM_h} t_j)$. It describes all 2-symbol words that match horizontally including the case of the right one belonging to the next row already. Then β_h^* describes all words s.t. between each even position and its right neighbour there is a correct matching. Hence, we can specify the correct horizontal tiling as follows.

$$\varphi_h \;\; := \;\; \texttt{Cl}(\; \beta_h^*; \top \cap \top; \beta_h^* \;)$$

The same trick cannot be used to axiomatise the tiling in vertical direction because we would have to list conjuncts for each position in the first row, i.e. exponentially many. This can easily be overcome using the $\texttt{G}$ operator.

$$\varphi_v \;\; := \;\; \texttt{G}\big(\texttt{Cl}(\; \beta_v^* \;)\big) \qquad \text{where} \quad \beta_v \;\; := \;\; \alpha_n \cap (\bigcup_{i=1}^{m} t_i; \top^*; (\bigcup_{j \in iM_v} t_j))$$

with α_n from the proof of Thm. 4. This includes some redundancy since the formula $\texttt{Cl}(\beta_v^*)$ – when interpreted in the cell (i,j) – checks for correct matchings between (i,j) and $(i,j+1)$, between $(i+1 \mod 2^n, j+1)$ and $(i+1 \mod 2^n, j+2)$, etc. The latter matchings are also imposed by the $\texttt{G}$ operator. □

Theorem 7. *Satisfiability in* $\text{TL}^{\text{RE}}[\texttt{Cl}]$ *is PSPACE-hard.*

Proof. By reduction from the tiling problem for $\{0,\ldots,n-1\} \times \mathbb{N}$, otherwise defined in the same way as the 2^n-tiling problem above. First note that for the corridor of width n no counter bits are needed because the vertical matching relation only requires statements of the form "in n steps" rather than 2^n. Here we assume a single proposition 0 marking the left edge of the corridor.

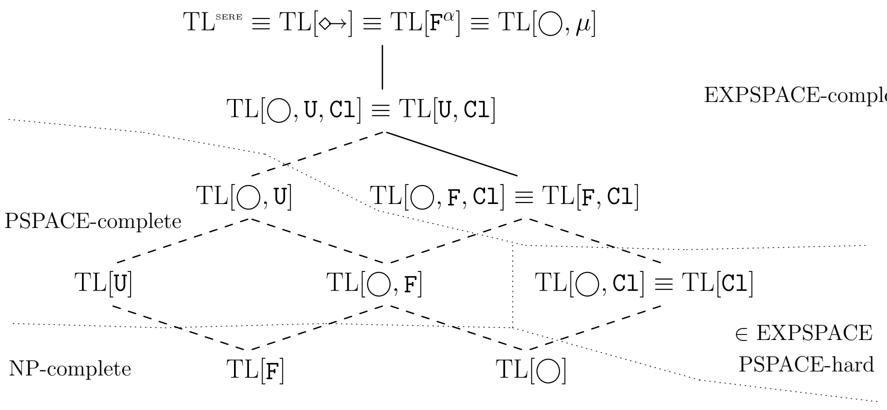
$$\varphi_{edge} \;\; := \;\; \texttt{Cl}(\; (0; \underbrace{\neg 0; \ldots; \neg 0}_{n-1 \text{ times}})^* \;)$$

Requiring every cell to carry a unique tile is done using φ_{tile} from the proof of Thm. 6. The horizontal matching relation can be axiomatised using formula φ_h from the proof of Thm. 6 with the proposition 0 instead of the formula χ_0. The vertical relation can be axiomatised as follows.

$$\varphi_v \;\; := \;\; \bigwedge_{i=0}^{n} \bigcirc^i \texttt{Cl}(\; (\bigcup_{i=1}^{m} t_i; \top^n; (\bigcup_{j \in iM_v} t_j))^* \;)$$

Note, again, that the first conjunct ensures matchings between $(0,0)$ and $(0,1)$, between $(1,1)$ and $(1,2)$, etc. The second conjunct ensures matchings between $(1,0)$ and $(1,1)$, between $(2,1)$ and $(2,2)$, etc. Therefore, we need $n+1$ conjuncts altogether to cover the entire corridor.

Let $\varphi := \varphi_{edge} \wedge \varphi_{tile} \wedge \varphi_h \wedge \varphi_v$. Finally, Lemma 3 shows that the $\bigcirc$-operators can be eliminated from φ at no blow-up which finishes the proof. □



$\mathrm{TL}^{\text{SERE}} \equiv \mathrm{TL}[\diamond\!\!\to] \equiv \mathrm{TL}[\mathtt{F}^\alpha] \equiv \mathrm{TL}[\bigcirc, \mu]$
$\mathrm{TL}[\bigcirc, \mathtt{U}, \mathtt{Cl}] \equiv \mathrm{TL}[\mathtt{U}, \mathtt{Cl}]$
EXPSPACE-complete
$\mathrm{TL}[\bigcirc, \mathtt{U}]$
$\mathrm{TL}[\bigcirc, \mathtt{F}, \mathtt{Cl}] \equiv \mathrm{TL}[\mathtt{F}, \mathtt{Cl}]$
PSPACE-complete
$\mathrm{TL}[\mathtt{U}]$
$\mathrm{TL}[\bigcirc, \mathtt{F}]$
$\mathrm{TL}[\bigcirc, \mathtt{Cl}] \equiv \mathrm{TL}[\mathtt{Cl}]$
∈ EXPSPACE
PSPACE-hard
NP-complete
$\mathrm{TL}[\mathtt{F}]$
$\mathrm{TL}[\bigcirc]$

Fig. 1. Expressive power and complexity of fragments of $\mathrm{TL}^{\text{SERE}}$

5 Summary and Conclusion

Fig. 1 shows the relationship between fragments of $\mathrm{TL}^{\text{SERE}}$ w.r.t. relative expressive power. Note that $\mathrm{TL}^{\text{SERE}}$ and the alike are equi-expressive to Monadic Second-Order Logic, resp. NBA or ω-regular expressions. LTL, i.e. $\mathrm{TL}[\bigcirc, \mathtt{U}]$, is equi-expressive to First-Order Logic or ω-star-free expressions.

Strict inclusions are marked using dashed lines. The strictness of these has been shown in [4] for the part below $\mathrm{TL}[\bigcirc, \mathtt{U}]$, and in Thm. 2 and Cor. 2 for the others. Strictness of the two remaining inclusions is still open.

Fig. 1 also gives an overview of the complexity of these fragments' satisfiability problems. Again, for the part below $\mathrm{TL}[\bigcirc, \mathtt{U}]$ this has been shown in [9] already. The other results summarise the findings of the theorems and corollaries in the previous section. The exact complexity of $\mathrm{TL}[\mathtt{Cl}]$ is still open. However, if the size of a (semi-extended) regular expression is measured as its syntactical length, then the problem becomes PSPACE-complete. Note that then the translation from $\mathrm{TL}[\mathtt{Cl}]$ formulas to $\exists\forall$-accepting NBA as shown in Lemma 2 produces NBA of at most exponential size whose emptiness can be checked in PSPACE again.

As for all linear time temporal logics, the satisfiability complexity results immediately carry over to the model checking problem for finite transition systems and the implicit "for all paths" semantics. The complexities are the same for all the fragments since the complexity classes mentioned there are deterministic – apart from the fragments $\mathrm{TL}[\mathtt{F}]$ and $\mathrm{TL}[\bigcirc]$. It is known that their model checking problems are co-NP-complete [9].

Apart from the open questions concerning the strictness of two inclusions, the work presented herein gives rise to some other questions regarding the expressive power and complexity of temporal logics with *extended regular expressions*, i.e. with a complementation operator included. It is known that its emptiness problem is non-elementary but it remains to be seen whether the presence of the temporal operators also lifts the satisfiability problem up the exponential space hierarchy by one level.

Note that in this setting the syntax of formulas contains a clear hierarchical structure: regular expressions can occur within temporal formulas but not vice-versa. It remains to be seen what the corresponding complexity and expressiveness results are when both kinds are allowed to be mixed in a straight-forward way: a temporal formula φ can be seen as an atomic proposition that holds in finite words of length exactly 1. Let $\text{TL}^*[..]$ denote the resulting fragments. With this mixture, it is easy to answer one of the open questions regarding expressive power: we have $\text{TL}[\bigcirc, \mathtt{U}] \leq \text{TL}^*[\mathtt{F}, \mathtt{Cl}]$ because of $\varphi \mathtt{U} \psi \equiv \mathtt{F}\, \psi \wedge \mathtt{Cl}((\varphi)^*; (\psi); \top^*)$.

Finally, a more thorough treatment of succinctness issues between these logics may be desirable.

References

1. Inc. Accellera Organization: Formal semantics of Accellera property specification language (2004) In Appendix B of http://www.eda.org/vfv/docs/PSL-v1.1.pdf
2. Barringer, H., Kuiper, R., Pnueli, A.: A really abstract concurrent model and its temporal logic. In: POPL'86. Conf. Record of the 13th Annual ACM Symp. on Principles of Programming Languages, pp. 173–183. ACM, New York (1986)
3. Bustan, D., Fisman, D., Havlicek, J.: Automata constructions for PSL. Technical Report MCS05-04, The Weizmann Institute of Science (2005)
4. Emerson, E.A., Halpern, J.Y.: Decision procedures and expressiveness in the temporal logic of branching time. Journal of Computer and System Sciences 30, 1–24 (1985)
5. Gabbay, D., Pnueli, A., Shelah, S., Stavi, J.: The temporal analysis of fairness. In: POPL'80. Proc. 7th Symp. on Principles of Programming Languages, pp. 163–173. ACM Press, New York (1980)
6. Henriksen, J.G., Thiagarajan, P.S.: Dynamic linear time temporal logic. Annals of Pure and Applied Logic 96(1–3), 187–207 (1999)
7. Pnueli, A.: The temporal logic of programs. In: FOCS'77. Proc. 18th Symp. on Foundations of Computer Science, Providence, RI, USA, pp. 46–57. IEEE Computer Society Press, Los Alamitos (1977)
8. Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. Journal of Computer and System Sciences 4, 177–192 (1970)
9. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. Journal of the Association for Computing Machinery 32(3), 733–749 (1985)
10. Sistla, A.P., Vardi, M.Y., Wolper, P.: Reasoning about infinite computation paths. In: FOCS'83. Proc. 24th Symp. on Foundations of Computer Science, pp. 185–194. IEEE Computer Society Press, Los Alamitos, CA, USA (1983)
11. Thomas, W.: Star-free regular sets of ω-sequences. Information and Control 42(2), 148–156 (1979)
12. van Emde Boas, P.: The convenience of tilings. In: Sorbi, A. (ed.) Complexity, Logic, and Recursion Theory. Lecture notes in pure and applied mathematics, vol. 187, pp. 331–363. Marcel Dekker, Inc. (1997)
13. Vardi, M.Y.: A temporal fixpoint calculus. In: ACM (ed.) POPL'88. Proc. Conf. on Principles of Programming Languages, pp. 250–259. ACM Press, NY, USA (1988)
14. Wolper, P.: Temporal logic can be more expressive. Information and Control 56, 72–99 (1983)

On Modal Refinement and Consistency

Kim G. Larsen, Ulrik Nyman, and Andrzej Wąsowski

Department of Computer Science, Aalborg University, Denmark
{kgl,ulrik,wasowski}@cs.aau.dk

Abstract. Almost 20 years after the original conception, we revisit several fundamental question about modal transition systems. First, we demonstrate the incompleteness of the standard modal refinement using a counterexample due to Hüttel. Deciding any refinement, complete with respect to the standard notions of implementation, is shown to be computationally hard (co-NP hard). Second, we consider four forms of consistency (existence of implementations) for modal specifications. We characterize each operationally, giving algorithms for deciding, and for synthesizing implementations, together with their complexities.

1 Background and Overview

Modal transition systems (MTSs) are a generalization of labeled transition systems (LTSs). Similarly to LTSs modal transition systems use labeled transitions between states to model behaviors. Unlike LTSs, they distinguish allowed and required behaviors (over- and under-approximations), which makes them a suitable semantic model for abstraction in program analysis and verification.

MTSs, originally introduced by Larsen and Thomsen almost 20 years ago [1], have since been applied in program analysis [2,3], model checking [4,5], verification [6,7], equation solving [8], interface theories [9], software product lines [9,10] and model merging [11,12]. Foundational work on modal transition systems included extensions to modal hybrid systems [13], timed modal specifications [14,15,16] and variants of disjunctive MTSs [8,17,18]. Surprisingly though, several fundamental questions about the theory of MTSs have never been addressed.

Refinement relations for modal transition systems are defined contravariantly. If S refines T then all allowed behaviors of S need to be allowed in T, while all required behaviors of T need also be required by S. An *implementation* is an MTS that has been completely specified, i.e. all its allowed behavior is also required, leaving no further choice for refinement. One fundamental issue for a modal refinement is to see whether it characterizes the inclusion of implementation sets thoroughly: can one for an MTS S refining an MTS T imply that all implementations of S are also implementations of T? And vice-versa?

Standard modal refinement is sound, but not complete in this sense. Meaning that here exist MTSs for which implementation inclusion holds, but which do not refine each other. We show that deciding any sound and complete refinement, preserving the set of implementations of standard modal refinement or weak modal refinement is co-NP hard. We conjecture the same for may-weak modal refinement [9] and branching refinement [10].

Modal transition systems of [1] are *syntactically consistent*, meaning that any required transition must also be allowed. This effectively disallows reasoning about inconsistencies, which is necessary for proper treatment of logical connectives in the context of modal transition systems (for example one would like to be able to express a modal transition system expressing a conjunction of two other MTSs that represent contradictory specifications). On the other hand, in [9], we have observed that other, more behavioral, notions of consistency might be useful. We have shown that systems that are *observationally consistent* with respect to some set of hidden actions, can be decomposed using parallel decomposition. We used this observation to build a product line theory in which modal transition systems play the role of behavioral variability models.

We believe that consistency should be decoupled from the basic definition of a modal transition system. In our opinion understanding a notion of consistency requires relating it to a notion of satisfiability, as typically done in logics. For example: a propositional formula is consistent if there exists a truth assignment on which the formula evaluates to true. In our context, modal transition systems play the role of formulæ, truth assignments are concrete implementations, and a refinement preorder is our satisfaction relation. Consequently, instead of proposing ad hoc *criteria* for consistency, we define consistency of a specification *semantically* as existence of a concrete implementation refining it.

Altogether we discuss four modal refinements and their induced consistencies. For each of these we define consistency semantically and find a computable criterion (a consistency relation) for deciding it. Then we study the complexity of consistency and the criterion. The results are summarized in Table 1.

Our choice of refinements and consistencies for this study is driven by existing work. We choose one known consistency (syntactic consistency) that have not been characterized using a refinement, and three known refinements (strong, may-weak and weak modal refinement) for which the related notions of consistency had never been formulated. However, we believe that consistency is not only of theoretical interest. Inconsistencies in specifications typically indicate modeling errors and thus procedures for detecting them find use in tools.

The contents of this paper are: the definition of modal transition systems and their refinement (Section 2), complexity analysis of completeness of this refinement (Section 3), a discussion of consistency notions induced by four modal refinements (Sections 4–7), a summary and a list of open problems (Section 8).

Table 1. Summary of consistency-related results

Modal refinement	Consistency	Lower bound	Upper bound	Section
syntactic	syntactic consistency [1]	linear time	linear time	4
strong [1]	strong consistency	NP-hard	exponential time	5
weak [19]	weak consistency	NP-hard	exponential time	6
may-weak [9]	may-weak consistency	NP-hard	exponential time	7

2 Modal Transition Systems

We introduce the basics following Larsen and Thomsen [1]. Assume a global set of actions *act* and write act^τ for $act \cup \{\tau\}$, where τ is a distinct internal action, such that $\tau \notin act$. A modal transition system is a triple $S = (states_S, \longrightarrow^S, \dashrightarrow^S)$, where $states_S$ is a set of states, also known as specifications [1] or processes. Then $\longrightarrow^S \subseteq states_S \times act^\tau \times states_S$ is a must-transition relation representing required transitions, and $\dashrightarrow^S \subseteq states_S \times act^\tau \times states_S$ is a may-transition relation representing allowed transitions.

In general the sets of states and transitions may be infinite, but we restrict ourselves to finite state systems with finite sets of actions in this paper. For simplicity we write $s \xrightarrow{a}^S s'$ iff $(s, a, s') \in \longrightarrow^S$, and $s \overset{a}{\dashrightarrow}^S s'$ iff $(s, a, s') \in \dashrightarrow^S$.

Larsen and Thomsen originally designed modal transition systems to be syntactically consistent meaning that all required transitions are also allowed: $\longrightarrow^S \subseteq \dashrightarrow^S$. Already in [14] Larsen lifts this restriction, with the argument that any sufficiently expressive specification language needs to be able to specify inconsistent specifications. This means that our transition systems are very much like mixed transition systems of Dams [20]. In Section 3 we follow the syntactic consistency requirement, while we relax it in later sections, generalizing the notion of consistency to strong and weak behavioral preorders. Regardless whether the consistency assumption is in place or not, we always separate the two transition relations explicitly to avoid confusion. A solid arrow represents just a must transition, without the possible related may transition. We draw both arrows when talking about a syntactically consistent must transition.

A modal transition system I is an *implementation* when the two transition relations coincide, $\longrightarrow^I = \dashrightarrow^I$. We use capital I to denote implementations and always state explicitly whenever a modal transition system is an implementation.

The following is the standard notion of strong refinement for modal transition systems introduced in [1] and generally accepted ever since:

Definition 1 (Modal Refinement). *For a pair of modal transition systems S and T a binary relation $\mathcal{R} \subseteq states_S \times states_T$ is a modal refinement between states of S and T iff for all $(s, t) \in \mathcal{R}$ and all actions a it holds that:*

for all $t' \in states_T$ such that $t \xrightarrow{a}^T t'$
  there exists an $s' \in states_S$ such that $s \xrightarrow{a}^S s'$ and $(s', t') \in \mathcal{R}$,
for all $s' \in states_S$ such that $s \overset{a}{\dashrightarrow}^S s'$
  there exists a $t' \in states_T$ such that $t \overset{a}{\dashrightarrow}^T t'$ and $(s', t') \in \mathcal{R}$.

We say that a state $s \in states_S$ refines a state $t \in states_T$, written $s \leq_{\mathrm{m}} t$, iff there exists a modal refinement containing (s, t).

If $\longrightarrow^T = \emptyset$ then this refinement collapses to regular simulation [21,22], while it coincides with bisimulation equivalence [23,24] if S and T are implementations.

3 Non-thoroughness of Modal Refinement

Already in the eighties there have been rumors of modal refinement being incomplete. However we were unable to find a published account of this fact, so we decided to include it here. We shall now define what we mean by completeness, proceeding to a counterexample witnessing the incompleteness of modal refinement. After this brief introduction we move to the first contribution of the paper: a discussion of the complexity class of a hypothetical complete refinement.

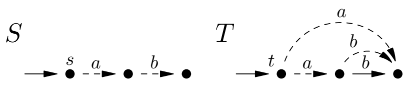


Fig. 1. $[\![S, s]\!] \subseteq [\![T, t]\!]$ and $s \not\leq_{\mathrm{m}} t$

For a state $s \in \mathit{states}_S$ let $[\![S, s]\!]$ denote the set of all its implementations such that $[\![S, s]\!] = \{(I, i) \mid i \leq_{\mathrm{m}} s \text{ and } \longrightarrow^{I} = \dashrightarrow^{I}\}$. Modal refinement is known to be sound, with respect to implementation inclusion: for $s \in \mathit{states}_S \wedge t \in \mathit{states}_T$, if $s \leq_{\mathrm{m}} t$ then also $[\![S, s]\!] \subseteq [\![T, t]\!]$, which follows directly from transitivity of $\leq_{\mathrm{m}}$. However $\leq_{\mathrm{m}}$ is not complete in this sense: there exist specifications S and T, with states s, t, such that $[\![S, s]\!] \subseteq [\![T, t]\!]$ but $s \not\leq_{\mathrm{m}} t$. This property of modal refinement is sometimes known as non-thoroughness [25]. Figure 1 presents a counterexample originating in the thesis of Hüttel [26, p. 32], also found in the thesis of Xinxin [27, p. 87] and in [18], albeit disguised in the context of disjunctive modal transition systems [8][1]. It contains two specifications S, T. It is a simple exercise to see that $[\![S, s]\!] = [\![T, t]\!]$, while $s \not\leq_{\mathrm{m}} t$.

3.1 A Thorough Refinement Is Co-NP Hard

Despite the non-thoroughness (incompleteness) of modal refinement its usefulness has never been questioned. This is probably because modal refinement is a natural generalization of both simulation and bisimulation and because it can be established efficiently (in time polynomial in the size of the transition systems). By showing that any complete refinement preserving precisely the same set of implementations as $\leq_{\mathrm{m}}$ cannot be decided in polynomial time (unless P=NP), we give yet another argument in favor of $\leq_{\mathrm{m}}$.

We show co-NP hardness by reducing 3-Dnf-Tautology to checking a sound and complete modal refinement in the above sense. Consider a propositional formula φ over n variables $x_1, \ldots, x_n$. It is clear that φ is a tautology iff $\mathit{true} \Rightarrow \varphi$ is a tautology. We will show how to construct, in polynomial time, a modal transition systems T_φ (representing a tautology over $x_1 \ldots x_n$) and S_φ (representing φ), so that $\mathit{true} \Rightarrow \varphi$ is a tautology iff $[\![T_\varphi, \mathit{true}]\!] \subseteq [\![S_\varphi, \varphi]\!]$, for selected initial states true and φ of T_φ and S_φ respectively. For simplicity we will assume that all clauses of φ are satisfiable. Satisfiability of a clause consisting of three conjunctions can be decided in constant time. Unsatisfiable clauses can thus be removed from φ in polynomial time, before we construct T_φ and S_φ. We choose the following states and actions for S_φ:

$$\mathit{states}_{S_\varphi} = \{\varphi, c_1, \ldots, c_m, \mathbf{0}\} \quad \{a, x_1, \ldots, x_n\} \subseteq \mathit{act} \ , \tag{1}$$

where c_i are clauses of φ, while $\mathbf{0}$ and a are fresh names.

First we explain how a single literal can be represented as a state with at most $n+1$ outgoing transitions. For a positive literal x_i we introduce a state x_i with a required transition $x_i \xrightarrow{x_i} \mathbf{0}$ and allowed transitions $x_i \overset{x_k}{\dashrightarrow} \mathbf{0}$ for all $k = 1 \,..\, n$.

[1] We thank Michael Huth, Harald Fecher, Heiko Schmidt and one of the anynonymous reviewers for helping to track down its origins.

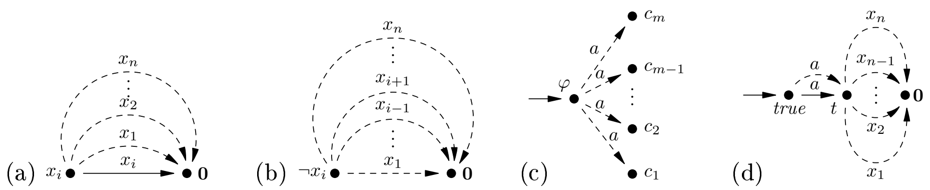


Fig. 2. Representing (a) a positive literal, (b) a negative literal, (c) a 3-DNF formula $\varphi = c_1 \vee \cdots \vee c_m$ and (d) a tautology over variables $x_1 \ldots x_n$

For a negative literal $\neg x_i$ we allow no outgoing must transitions and create may transitions $(\neg x_i) \overset{x_k}{\dashrightarrow} \mathbf{0}$ for all $k \neq i$. Positive assignments are represented by must transitions, and negative assignments are represented by lack of may transitions. Assignments with no effect on satisfaction of the formula are modeled by may transitions with no corresponding must transitions. See Figure 2ab.

Now generalize this to conjunctive clauses of a 3-DNF formula. A clause $l_1 \wedge l_2 \wedge l_3$ is translated into a state labeled $l_1 \wedge l_2 \wedge l_3$ with the following transitions:

1° $(l_1 \wedge l_2 \wedge l_3) \xrightarrow{x_i}{}^{S_\varphi} \mathbf{0}$ iff $l_k = x_i$ for some $k = 1..3$.
2° $(l_1 \wedge l_2 \wedge l_3) \overset{x_i}{\dashrightarrow}{}^{S_\varphi} \mathbf{0}$ iff $l_k \neq \neg x_i$ for all $k = 1..3$.

Since we only consider satisfiable clauses, modal transition systems created this way are syntactically consistent (all required transitions are allowed). A satisfying truth assignment to $l_1 \wedge l_2 \wedge l_3$ can be extracted from any implementation I refining the state with the same label—just set x_i to *true* iff $I \xrightarrow{x_i}$ and set x_i to *false* otherwise. Similarly we can construct an implementation refining $l_1 \wedge l_2 \wedge l_3$ given any satisfying assignment to this clause.

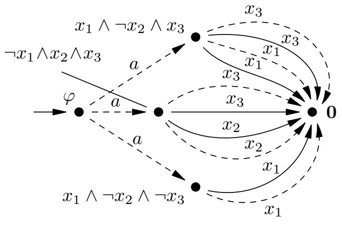

$x_1 \wedge \neg x_2 \wedge x_3$
$\neg x_1 \wedge x_2 \wedge x_3$
$x_1 \wedge \neg x_2 \wedge \neg x_3$
φ
a
a
a
x_3
x_3
x_1
x_1
x_3
x_3
x_2
x_2
x_1
x_1
0

Fig. 3. Reduction for $\varphi = (x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge \neg x_3)$

A 3-DNF formula $\varphi = c_1 \vee \ldots \vee c_m$ is represented using a state labeled φ and may transitions to its clauses: $\varphi \overset{a}{\dashrightarrow}{}^{S_\varphi} c_i$ for $i = 1..m$. No must transitions are generated. See Figure 2c and 3. States labeled c_i represent processes resulting from translation of the individual clauses as presented above.

Observe that each satisfying assignment to formula φ has a corresponding deterministic implementation of S_φ. Also each implementation of S_φ embeds at most one satisfying assignment to φ extracted using the same rules as discussed for clauses (one per each nondeterministic choice in the initial state of the implementation). Clearly S_φ can be constructed in time polynomial in the size of φ.

We now consider construction of T_φ. First let $states_{T_\varphi} = \{true, t_\varphi, \mathbf{0}\}$. We also create the following transitions: $true \xrightarrow{a}{}^{T_\varphi} t_\varphi$, $true \overset{a}{\dashrightarrow}{}^{T_\varphi} t_\varphi$, and $t_\varphi \overset{x_i}{\dashrightarrow}{}^{T_\varphi} \mathbf{0}$ for all variables x_i of φ (See Fig. 2d). Clearly T_φ can be constructed in time at most polynomial in size of φ.

The following lemma states the correctness of our reduction.

Lemma 2. *A 3-DNF formula φ with all satisfiable clauses is a tautology iff $[\![T_\varphi, \mathit{true}]\!] \subseteq [\![S_\varphi, \varphi]\!]$.*

Proof. We first consider the direction right to left, i.e. assume that $[\![T_\varphi, \mathit{true}]\!] \subseteq [\![S_\varphi, \varphi]\!]$ and take any truth assignment ϱ to variables x_i of φ. We construct a deterministic implementation I_ϱ in the following way: $\mathit{states}_{I_\varrho} = \{t, \varrho, \mathbf{0}\}$, where there are two transition from t to ϱ: $t \xrightarrow{a} \varrho \wedge t \overset{a}{\dashrightarrow} \varrho$ and for all x_i such that $\varrho(x_i) = \mathit{true}$: $\varrho \xrightarrow{x_i} \mathbf{0} \wedge \varrho \overset{x_i}{\dashrightarrow} \mathbf{0}$. Due to the construction of our reduction this means that ϱ satisfies φ. Since for any assignment ϱ we can conclude that φ holds, φ is a tautology.

Now consider the claim of the lemma from left to right. We address its contrapositive. Assume that there exists an implementation I and its state t such that $t \leq_{\mathrm{m}} \mathit{true}$, but $t \not\leq_{\mathrm{m}} \varphi$. We want to show that φ is not a tautology. Observe that since $t \not\leq_{\mathrm{m}} \varphi$ there must exist a state $s \in \mathit{states}_I$ such that $t \xrightarrow{a} s$ and for all clause states c_i of S_φ it is the case that $s \not\leq_{\mathrm{m}} c_i$. But this means that the assignment represented by s (present x_i-transitions give rise to $x_i = \mathit{true}$, absent to $x_i = \mathit{false}$) falsifies φ meaning that φ is not a tautology. □

Theorem 3. *The problem of deciding $[\![T, t]\!] \subseteq [\![S, s]\!]$ for states t and s of arbitrary modal transition systems T and S respectively is co-NP hard.*

Co-NP hardness follows from the above reduction and co-NP hardness of 3-DNF-Tautology. The same reduction can be used to show that the thorough refinement induced by weak modal refinement (Section 6) is also co-NP hard to decide. We omit that proof as the argument is rather similar to the above.

4 Syntactic Consistency and Syntactic Refinement

From now on we relax the syntactic consistency requirement presented in Section 2, and allow reasoning about systems for which $\longrightarrow \not\subseteq \dashrightarrow$. We will introduce a syntactic refinement $\subseteq_m$ with its induced notion of consistency and prove that it is (almost) precisely characterized by the syntactic consistency. These results are very simple, but we include them for three reasons. First, we cannot avoid discussing the most well known notion of consistency for modal transition systems (a notion that had never been characterized using a refinement relation). Second, we can show a refinement inducing this consistency (a refinement that had never been explicitly linked to any consistency notion). Third, we want to present all ingredients of a consistency study using a simple example: a refinement, its induced consistency, operational characterization in form of a consistency relation, and a coincidence proof. Later sections will follow exactly the same pattern.

Definition 4 (Syntactic Refinement). *For two modal transition systems S and T a syntactic refinement $\mathcal{R}$ is a partial injective function on states_S into states_T such that for all pairs (s, t), $t = \mathcal{R}(s)$, and all actions a it holds that*

for all $t' \in states_T$ such that $t \xrightarrow{a}^T t'$
there exists an $s' \in states_S$ such that $s \xrightarrow{a}^S s'$ and $t' = \mathcal{R}(s')$,
for all $s' \in states_S$ such that $s \overset{a}{\dashrightarrow}^S s'$
there exists a $t' \in states_T$ such that $t \overset{a}{\dashrightarrow}^T t'$ and $t' = \mathcal{R}(s')$.

A state s is said to be a syntactic refinement of a state t, written $s \subseteq_m t$, if there exists a syntactic refinement function $\mathcal{R}$ such that $t = \mathcal{R}(s)$.

Intuitively this refinement establishes that the may-transition graph of S is a subgraph of the may-transition graph of T and that the must-transition graph of T is a subgraph of the must-transition graph of S.

Definition 5 (Syntactic Consistency). *A state $s \in states_S$ is syntactically consistent iff there exists an implementation I and its state s^I such that $s^I \subseteq_m s$.*

We claim that this notion of semantic consistency (almost) coincides with the one presented in Section 2. For the sake of uniformity let us reformulate that definition using an explicit notion of consistency relation:

Definition 6 (Syntactic Consistency Relation). *Given a modal transition system S, a binary relation $\mathcal{S} \subseteq states_S \times states_S$ is a syntactic consistency relation on states of S iff for each state s if $(s, s) \in \mathcal{S}$ and each action $a \in act$ it holds that whenever $s \xrightarrow{a} s'$ for some $s' \in states_S$ then also $s \overset{a}{\dashrightarrow} s'$ and $(s', s') \in \mathcal{S}$.*

For a syntactic consistency relation $\mathcal{S}$ and a state $s \in states_S$ such that $(s, s) \in \mathcal{S}$, we synthesize an implementation $I_{\mathcal{S}}$ with a state s^I such that $s^I \subseteq_m s$. Take states of $I_{\mathcal{S}}$ to be consistent states of S: $states_{I_{\mathcal{S}}} = \{p \in states_S \mid (p, p) \in \mathcal{S}\}$ and $s^I = s$. The transition relation of $I_{\mathcal{S}}$ is the must transition relation of S projected on states of $I_{\mathcal{S}}$: $\longrightarrow^{I_{\mathcal{S}}} = \dashrightarrow^{I_{\mathcal{S}}} = \longrightarrow^S \cap (states_{I_{\mathcal{S}}} \times act^{\tau} \times states_{I_{\mathcal{S}}})$.

Theorem 7 (Soundness). *If there exists a syntactic consistency relation containing a state s of S then s is a syntactically consistent state in the sense of Definition 5. Moreover the implementation $I_{\mathcal{S}}$ constructed above is one of its refinements: $s^I \leq_{\mathrm{m}} s$.*

It turns out that syntactic consistency relations characterize syntactic consistency in the sense of Definition 5 in a complete manner. Given a syntactic implementation I of a modal transition system S ($I \subseteq_m S$) we can construct a syntactic consistency relation in the following way:

$$\mathcal{S}_I = \{(q, q) \in states_S \mid \text{exists } p \in states_I \wedge p \subseteq_m q\} \tag{2}$$

Theorem 8 (Completeness). *Let s be a state of a modal transition system S and s^I be a state of an implementation I such that $s^I \subseteq_m s$. Then there exists a syntactic consistency relation for S containing (s, s), and $\mathcal{S}_I$ is one of such.*

Since establishing consistency of models is a useful feature in modeling tools, we remark that the cost of deciding existence of syntactic implementations (via consistency relations) for a state $s \in states_S$ is at most (and at least) linear in the size of S. The algorithm corresponds to a traversal of the must-transition graph starting in s, and checking the consistency requirement in each state.

Syntactic consistency relations characterize syntactic consistency in the sense of [1] *almost* precisely. In fact the two notions coincide if all states of S are reachable from s via must transitions. Otherwise Definitions 5 and 6 allow inconsistencies in unreachable parts, which has not been taken into account in [1].

5 Strong Modal Refinement and Strong Consistency

In Section 2 we have recalled the notion of (strong) modal refinement. Now we introduce its induced notion of consistency and characterize it operationally.

Definition 9 (Strong Consistency). *A state s of a modal transition system S is strongly consistent iff there exists an implementation I and its state s^I such that $s^I \leq_{\mathrm{m}} s$.*

In order to give an operational characterization of strong consistency we need to lift the transition relations to sets of states. For sets $\sigma, \sigma' \subseteq \mathit{states}_S$ we write:

$$\sigma \xrightarrow{a}^{\lfloor S \rfloor} \sigma' \quad \text{iff} \quad \exists s \in \sigma.\, \exists s' \in \sigma'.\, s \xrightarrow{a}^{S} s' \ , \tag{3}$$

$$\sigma \overset{a}{\dashrightarrow}^{\lfloor S \rfloor} \sigma' \quad \text{iff} \quad \forall s \in \sigma.\, \exists s' \in \sigma'.\, s \overset{a}{\dashrightarrow}^{S} s' \ . \tag{4}$$

Definition 10 (Strong Consistency Relation). *Given a modal transition system S, a relation $\mathcal{B} \subseteq \mathcal{P}(\mathit{states}_S)$ is a strong consistency relation on states_S iff for all actions $a \in \mathit{act}$ and all $\sigma \in \mathcal{B}$ the following condition is satisfied:*

whenever $s \overset{a}{\dashrightarrow}^{S} s'$ for some $s \in \sigma$ and some $s' \in \mathit{states}_S$
then also $\sigma \xrightarrow{a}^{\lfloor S \rfloor} \sigma'$ and $\sigma \overset{a}{\dashrightarrow}^{\lfloor S \rfloor} \sigma'$ for some $\sigma' \in \mathcal{B}$ containing s'.

Elements of $\mathcal{B}$ are called consistency classes. $\mathcal{B}$ is a strong consistency relation for a state $s \in \mathit{states}_S$ iff it contains a consistency class σ_s such that $s \in \sigma_s$.

Given a consistency relation $\mathcal{B}$ for a state $s \in \mathit{states}_S$ we can synthesize an implementation $I_{\mathcal{B}}$ with a state $s^I \in \mathit{states}_{I_{\mathcal{B}}}$, such that $s^I \leq_{\mathrm{m}} s$. Take the consistency classes of $\mathcal{B}$, to be the states of $I_{\mathcal{B}}$: $\mathit{states}_{I_{\mathcal{B}}} = \mathcal{B}$ and s^I be the class σ_s containing s. Both transition relations of $I_{\mathcal{B}}$ equal the intersection of *must* and *may* transition relations of S lifted to consistency classes of $\mathcal{B}$:

$$\sigma \xrightarrow{a}^{I_{\mathcal{B}}} \sigma' \text{ and } \sigma \overset{a}{\dashrightarrow}^{I_{\mathcal{B}}} \sigma' \quad \text{iff} \quad \sigma \xrightarrow{a}^{\lfloor S \rfloor} \sigma' \text{ and } \sigma \overset{a}{\dashrightarrow}^{\lfloor S \rfloor} \sigma' \ . \tag{5}$$

Theorem 11 (Soundness). *If there exists a consistency relation $\mathcal{B}$ for a modal transition system S then S is strongly consistent in the sense of Definition 9. Moreover $I_{\mathcal{B}}$ constructed as above is one of its refinements: $s^I \leq_{\mathrm{m}} S$.*

Strong consistency relations characterize strong consistency in a sound and complete manner. Given a state s^I of an implementation I refining a state $s \in \mathit{states}_S$ ($s^I \leq_{\mathrm{m}} s$) we can construct a consistency relation $\mathcal{B}_I$ for S following (6):

$$\mathcal{B}_I = \{\sigma_p \subseteq \mathit{states}_S \mid p \in \mathit{states}_I \text{ and } \sigma_p \neq \emptyset \text{ and } \forall q \in \sigma_p.\, p \leq_{\mathrm{m}} q\} \tag{6}$$

Observe that the σ_p sets above are not necessarily maximal.

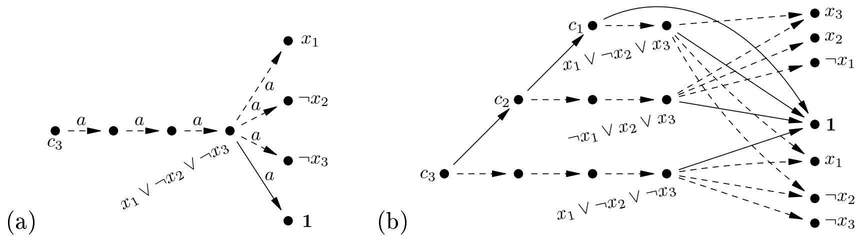


Fig. 4. Representing (a) a disjunctive clause and (b) a translation for φ

Theorem 12 (Completeness). *Let $s \in states_S$ and let I be an implementation, let $s^I \in states_I$ and $s^I \leq_{\mathrm{m}} s$. Then there exists a consistency relation for the state s. Also relation $\mathcal{B}_I$ defined above is one of such relations.*

Definition 10 can be interpreted operationally giving a simple exponential fixpoint algorithm: start with a singleton class containing s and apply the rule generating classes until a fixpoint is reached.

We demonstrate that the problem of deciding strong consistency is in fact NP-hard using a reduction from 3-Cnf-Sat. Let $\varphi = c_1 \wedge \ldots \wedge c_m$ be a 3-CNF formula over variables $x_1, \ldots, x_n$. Construct a modal transition system S_φ such that its state labeled c_m is consistent iff φ is satisfiable. The states of S_φ are literals of φ, a $\mathbf{0}$ state, a $\mathbf{1}$ state (a state allowing any behavior: $\mathbf{1}\overset{x_i}{\dashrightarrow}{}^{S_\varphi}\mathbf{1}$ for all $i = 1..n$ and $\mathbf{1}\overset{a}{\dashrightarrow}{}^{S_\varphi}\mathbf{1}$), plus a polynomial number of auxiliary states. We shall use an action per each variable x_i and one auxiliary action a.

Literals in φ are translated to states using the principle shown in Figure 2ab. A disjunction of three literals $l_1 \vee l_2 \vee l_3$ is represented by a state labeled $(l_1 \vee l_2 \vee l_3)$ such that $(l_1 \vee l_2 \vee l_3)\xrightarrow{a}{}^{S_\varphi}\mathbf{1}$ and $(l_1 \vee l_2 \vee l_3)\overset{a}{\dashrightarrow}{}^{S_\varphi} l_k$ for all $k = 1..3$. Now each clause c_i is represented by a state labeled c_i followed by a sequence of exactly i may a-transitions leading to the state representing the disjunction. For regularity we assume that there is a special *true* clause c_0, that we translate to $\mathbf{1}$. Figure 4a shows the result of translating a clause $c_3 = x_1 \vee \neg x_2 \vee \neg x_3$. Recall that states labeled with literals are actually results of translation of Figure 2ab.

Now the top-level conjunction is translated inductively. First representations of $c_1, \ldots, c_m$ are created as above, then they are conjoined using must transitions. The ith clause is conjoined by a must transition from c_i to c_{i-1}: $c_i \xrightarrow{a}{}^{S_\varphi} c_{i-1}$. Note that we add at most a quadratic number of auxiliary states this way (and a similar number of transitions). After conjoining c_m we obtain a representation of the whole formula. Figure 4b presents a complete translation for a formula $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$. All unlabeled transitions should actually be labeled by a (removed to decrease clutter).

It is not hard to see that if the c_m state has an implementation then it actually has a state that satisfies the requirements of all the states representing disjunctions, and thus it induces a satisfiable assignment to φ.

6 Weak Refinement and Weak Consistency

We shall now discuss what is considered a classic form of a weak modal refinement (obtained by transforming modal refinement in the same way as bisimulation is transformed in order to obtain its weak form; to the best of our knowledge first published by Hüttel and Larsen in [19]). The definition uses a notion of weak transition relations that we introduce first. We shall write:

$$s \xrightarrow{a}_{*}^{S} s' \quad \text{iff} \quad s\,(\xrightarrow{\tau}^{S})^{*} \xrightarrow{a}^{S} (\xrightarrow{\tau}^{S})^{*}\, s' \tag{7}$$

$$s \dashrightarrow^{a}_{*}{}^{S} s' \quad \text{iff} \quad s\,(\dashrightarrow^{\tau}{}^{S})^{*} \dashrightarrow^{a}{}^{S} (\dashrightarrow^{\tau}{}^{S})^{*}\, s'\ , \tag{8}$$

where $\mathcal{R}^*$ denotes zero or more transitive applications of a binary relation $\mathcal{R}$. Finally we write $s \xrightarrow{\hat{a}}_{*}^{S} s'$ whenever $s \xrightarrow{a}_{*}^{S} s'$ and $a \neq \tau$, or whenever $s\,(\xrightarrow{\tau}^{S})^{*}\, s'$ and $a = \tau$. Similarly for the may transition relation.

Definition 13 (Weak Modal Refinement). *Let S, T be modal transition systems. A binary relation $\mathcal{R} \subseteq states_S \times states_T$ is a weak modal refinement iff for each pair $(s,t) \in \mathcal{R}$ and each action $a \in act^{\tau}$ it holds that:*

for all $t' \in states_T$ such that $t \xrightarrow{a}^{T} t'$
there exists $s' \in states_S$ such that $s \xrightarrow{\hat{a}}_{*}^{S} s'$ and $(s',t') \in \mathcal{R}$,
for all $s' \in states_S$ such that $s \dashrightarrow^{a}{}^{S} s'$
there exists $t' \in states_T$ such that $t \dashrightarrow^{\hat{a}}_{*}{}^{T} t'$ and $(s',t') \in \mathcal{R}$.

We say that a state $s \in states_S$ weakly refines a state $t \in states_T$, written $s \leq_{\mathrm{m}}^{*} t$ iff there exists a weak modal refinement containing (s,t).

Definition 14 (Weak Consistency). *A state s of a modal transition system S is weakly consistent iff there exists an implementation I and its state s^I such that $s^I \leq_{\mathrm{m}}^{*} s$.*

We characterize weak consistency using consistency relations as before. In order to do this we need to lift weak transition relations $\dashrightarrow_{*}$ and $\longrightarrow_{*}$ to sets of states. For two sets of states $\sigma, \sigma' \subseteq states_S$ write:

$$\sigma \xrightarrow{\hat{a}}_{*}^{\lfloor S \rfloor} \sigma' \quad \text{iff} \quad \exists s \in \sigma.\, \exists s' \in \sigma'.\, s \xrightarrow{\hat{a}}_{*}^{S} s'\ , \tag{9}$$

$$\sigma \dashrightarrow^{\hat{a}}_{*}{}^{\lfloor S \rfloor} \sigma' \quad \text{iff} \quad \forall s \in \sigma.\, \exists s' \in \sigma'.\, s \dashrightarrow^{\hat{a}}_{*}{}^{S} s'\ . \tag{10}$$

Definition 15 (Weak Consistency Relation). *Let S be a modal transition system. A relation $\mathcal{O} \subseteq \mathcal{P}(states_S)$ is a weak consistency relation on $states_S$ iff for any set $\sigma \in \mathcal{O}$, for any state $s \in \sigma$, and for any action $a \in act^{\tau}$ it holds that:*

whenever $s \xrightarrow{a}^{S} s'$ for some $s' \in states_S$
then also $\sigma \xrightarrow{\hat{a}}_{*}^{\lfloor S \rfloor} \sigma'$ and $\sigma \dashrightarrow^{\hat{a}}_{*}{}^{\lfloor S \rfloor} \sigma'$ for some $\sigma' \in \mathcal{O}$ containing s'.

$\mathcal{O}$ is a weak consistency relation for a state $s \in states_S$ iff it contains a consistency class σ_s such that $start_s \in \sigma_s$.

As before, we claim that weak consistency relations (Definition 15) soundly characterize weak consistency (Definition 14): for a state $s \in \mathit{states}_S$ with a known weak consistency relation $\mathcal{O}$, one can construct a weak implementation $I_{\mathcal{O}}$ containing a state s^I such that $s^I \leq_{\mathrm{m}}^* s$. Take states of $I_{\mathcal{O}}$ to be consistency classes of $\mathcal{O}$ ($\mathit{states}_{I_{\mathcal{O}}} = \mathcal{O}$), and s^I be a class σ_s containing s. The transition relations of $I_{\mathcal{O}}$ are the intersection of the weak transition relations of S lifted to consistency classes of $\mathcal{O}$. For all actions $a \in \mathit{act}^\tau$:

$$\sigma \xrightarrow{a}{}^{I_{\mathcal{O}}} \sigma' \text{ and } \sigma \overset{a}{\dashrightarrow}{}^{I_{\mathcal{O}}} \sigma' \quad \text{iff} \quad \sigma \xrightarrow{\hat{a}}{}_*^{\lfloor S \rfloor} \sigma' \text{ and } \sigma \overset{\hat{a}}{\dashrightarrow}{}_*^{\lfloor S \rfloor} \sigma' \ . \tag{11}$$

Theorem 16 (Soundness). *Let S be a modal transition system, $s \in \mathit{states}_S$, and $\mathcal{O}$ be a weak consistency relation for s. Then s is weakly consistent and $s^I \in \mathit{states}_{I_{\mathcal{O}}}$ is one of its implementations: $s^I \leq_{\mathrm{m}}^* s$.*

Consistency relations characterize weak consistency precisely. Assume that a state $s \in \mathit{states}_S$ is refined by a state s^I of an implementation I ($I \leq_{\mathrm{m}}^* S$). Then one can use this implementation to construct the consistency relation $\mathcal{O}_I$:

$$\mathcal{O}_I = \{\sigma_p \subseteq \mathit{states}_S \mid p \in \mathit{states}_I \text{ and } \sigma_p \neq \emptyset \text{ and } \forall q \in \sigma_p . p \leq_{\mathrm{m}}^* q\} \tag{12}$$

Theorem 17 (Completeness). *Let S be a modal transition system, I be an implementation, and let $s^I \leq_{\mathrm{m}}^* s$ for some $s^I \in \mathit{states}_I$ and $s \in \mathit{states}_S$. Then there exist weak consistency relations for s, and $\mathcal{O}_I$ is one of them.*

Definition 15 can be interpreted operationally giving rise to an exponential algorithm for constructing a consistency relation and deciding weak consistency. Weak consistency collapses to strong consistency for systems without transitions labeled with τ. Consequently the problem of deciding it is at least NP-hard, by reduction from 3-Cnf-Sat presented in Section 5.

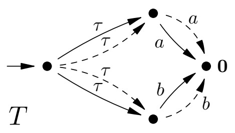


Fig. 5. All implementations of T have τ-transitions

We conclude this section with a comment on synthesis of a weak implementation $I_{\mathcal{O}}$ from a consistency relation $\mathcal{O}$. The implementation synthesized by the algorithm presented above will contain internal transitions, if the specification contained them. In fact this is not always necessary—there definitely exist specifications with internal transitions that can be realized without hidden behavior. However, hidden transitions are unavoidable for some specifications. Figure 5 shows such a specification (in fact even a syntactically consistent one).

7 May-Weak Modal Refinement and Its Consistency

In [9] we have proposed another weakening of modal refinement, generalizing alternating simulation [28] for two players as used in interface automata [29]. We call it *may-weak* here, as it preserves strong behavior on must transitions, only allowing weak matching on may transitions. It has been demonstrated that may-weak modal refinement is a sound basis for assume/guarantee reasoning: it preserves absence of deadlocks on guaranteed behaviors (details in [9]).

Before we can define the may-weak refinement, let us define the may-weak transition relation as used in this refinement. We shall write

$$s \overset{a}{\dashrightarrow}{}_{\triangleleft}^{S} s' \quad \text{iff} \quad s(\overset{\tau}{\dashrightarrow}{}^{S})^{*} s'' \overset{a}{\dashrightarrow}{}^{S} s' \tag{13}$$

Similarly as before we write $s \overset{\hat{a}}{\dashrightarrow}{}_{\triangleleft}^{S} s'$ meaning $s \overset{a}{\dashrightarrow}{}_{\triangleleft}^{S} s'$ if $a \in \mathit{act}$ and $s(\overset{\tau}{\dashrightarrow}{}^{S})^{*} s'$ if $a = \tau$. We use the regular (strong) must-transition relation lifted to sets of states as in Section 5. We also lift our new may-weak transition relation:

$$\sigma \overset{\hat{a}}{\dashrightarrow}{}_{\triangleleft}^{\lfloor S \rfloor} \sigma' \text{ iff } \forall s \in \sigma . \exists s' \in \sigma' .\, s \overset{\hat{a}}{\dashrightarrow}{}_{\triangleleft}^{S} s' \ . \tag{14}$$

Let us now define may-weak modal refinement [9] using may-weak transitions:

Definition 18 (May-weak Modal Refinement). *A binary relation $\mathcal{R} \subseteq \mathit{states}_S \times \mathit{states}_T$ is a may-weak refinement between states of two modal transition systems S and T iff for each pair of states $(s,t) \in \mathcal{R}$ it holds that:*

for all $a \in \mathit{act}$ and for all $t' \in \mathit{states}_T$ such that $t \xrightarrow{a}{}^{T} t'$
there exists $s' \in \mathit{states}_S$ such that $s \xrightarrow{a}{}^{S} s'$ and $(s',t') \in \mathcal{R}$,
for all $a \in \mathit{act}^{\tau}$ and for all $s' \in \mathit{states}_S$ $s \overset{a}{\dashrightarrow}{}^{S} s'$
there exists $t' \in \mathit{states}_{T'}$ such that $t \overset{\hat{a}}{\dashrightarrow}{}_{\triangleleft}^{T} t'$ and $(s',t') \in \mathcal{R}$.

A state $s \in \mathit{states}_S$ may-weakly refines a state $t \in \mathit{states}_T$, written $s \leq_{\mathrm{m}}^{\triangleleft} t$ iff there exists a may-weak modal refinement containing (s,t).

Definition 19 (May-weak Consistency). *A state s of a modal transition system S is may-weak consistent iff there exists an implementation I and its state s^I such that $s^I \leq_{\mathrm{m}}^{\triangleleft} s$.*

Definition 20 (May-weak Consistency Relation). *Let S be a modal transition system. A relation $\mathcal{U} \subseteq \mathcal{P}(\mathit{states}_S)$ is a may-weak consistency relation on states_S iff for any set of states $\sigma \in \mathcal{U}$, for any state $s \in \sigma$, and for any action $a \in \mathit{act}$ the following holds:*

whenever $s \xrightarrow{a}{}^{S} s'$ for some $s' \in \mathit{states}_S$
then also $\sigma \xrightarrow{a}{}^{\lfloor S \rfloor} \sigma'$ and $\sigma \overset{a}{\dashrightarrow}{}_{\triangleleft}^{\lfloor S \rfloor} \sigma'$ for some $\sigma' \in \mathcal{U}$ containing s'.

$\mathcal{U}$ is a may-weak consistency relation for a state $s \in \mathit{states}_S$ iff it contains a consistency class $\sigma_s \in \mathcal{U}$ such that $s \in \sigma_s$.

Given a consistency relation $\mathcal{U}$ for a state s of a modal transition system S, we can synthesize an implementation $I_{\mathcal{U}}$ with a state s^I refining s. The states of $I_{\mathcal{U}}$ are the consistency classes of $\mathcal{U}$: $\mathit{states}_{I_{\mathcal{U}}} = \mathcal{U}$ and s^I is the consistency class containing s. Transition relations of $I_{\mathcal{U}}$ equal intersection of *must* and may-weak transition relations of S lifted to consistency classes in $\mathcal{U}$ (for $a \neq \tau$):

$$\sigma \xrightarrow{a}{}^{I_{\mathcal{U}}} \sigma' \text{ and } \sigma \overset{a}{\dashrightarrow}{}^{I_{\mathcal{U}}} \sigma' \quad \text{iff} \quad \sigma \xrightarrow{a}{}^{\lfloor S \rfloor} \sigma' \text{ and } \sigma \overset{a}{\dashrightarrow}{}_{\triangleleft}^{\lfloor S \rfloor} \sigma' \ , \tag{15}$$

Theorem 21 (Soundness). *Let $s \in states_S$. If $\mathcal{U}$ is a may-weak consistency relation for s then s is may-weakly consistent and $s^I \in states_{I_\mathcal{U}}$ constructed as above is one of its implementations: $s^I \leq_{\mathrm{m}}^{\lhd} s$.*

For the completeness of characterization consider an implementation I, a state $s^I \in states_I$ such that $s^I \leq_{\mathrm{m}}^{\lhd} s$, where $s \in states_S$. We construct a consistency relation $\mathcal{U}_I$ for s in the following way:

$$\mathcal{U}_I = \{\sigma_p \subseteq states_S \mid p \in states_I \text{ and } \sigma_p \neq \emptyset \text{ and } \forall q \in \sigma_p .\, p \leq_{\mathrm{m}}^{\lhd} q\} \ . \qquad (16)$$

Theorem 22 (Completeness). *Let S be a modal transition system, $s \in states_S$ and let I be an implementation such that $s^I \leq_{\mathrm{m}}^{\lhd} s$ for some $s^I \in states_I$. Then there exist a may-weak consistency relation for s, and $\mathcal{U}_I$ is one such relation.*

Existence of a may-weak consistency relation for a given state s can be decided in exponential time, using an algorithm that is easy to extract from Definition 20. As previously this problem is also NP-hard, as may-weak consistency collapses to strong consistency for specifications without τ transitions.

A remarkable property of may-weak modal refinement, which we have not realized when writing [9], is that a may-weak consistent system always has implementations that contain no hidden actions ($I_\mathcal{U}$ above is actually constructed without introducing internal transitions). This is because this refinement captures a kind of (observation) determinism of required behaviors in specifications. We find this property appealing for applications again: it describes a class of specifications which allow implementations that are predictable (provided that they are deterministic). As predictability is an important property of software systems, the above decision procedure is likely to prove useful in practice.

8 Conclusion and Open Problems

We have addressed several basic questions in the theory of modal transition systems. We have shown that deciding any refinement that captures, in a precise way, the same set of concrete implementations as the standard modal refinement (or weak modal refinement) is co-NP hard. This lower bound is not tight. An upper bound of EXPTIME is easily established by casting the problem as checking satisfiability of implication between two characteristic formulas, in the modal μ-calculus. Finding a tight bound remains an open problem that we shall address shortly. We also hope to study hardness of thorough refinements induced by may-weak modal refinement and branching modal refinement [10].

Furthermore we have contributed to the understanding of the relation between refinements and consistencies studying notions of consistency for modal transition systems induced by four different refinement relations: syntactic consistency [1] (induced by a graph inclusion refinement), strong consistency (induced by a regular modal refinement [1]), weak consistency (induced by weak modal refinement [19]) and may-weak consistency (induced by may-weak modal refinement [9]). For each of these we have given a sound and complete operational characterization. The upper bound on establishing the last three of these consistencies is exponential, and they are NP-hard. Syntactic consistency can be established in linear time.

There is a range of open problems related to these results. First, it is an interesting question whether there exists a useful alternative to modal refinement that completely characterizes its own (as opposed to the currently accepted) set of implementations and that can be decided in polynomial time. The main challenge here is to argue that the set of implementations considered is interesting from a practical point of view. Alternatively, as suggested to us by Michael Huth, one can try to characterize broad classes of modal transition systems for which the currently used refinement is complete.

Finding a uniform formulation for four consistency studies as presented in this paper was a rather challenging but rewarding task. Given that they can be described so similarly one could try to take this analogy further and design a more abstract meta-consistency theory, parameterized only by a refinement.

Furthermore it is interesting to study the relation between consistency and parallel decomposition. We have done some preliminary work on that topic in [9], though in a rather restricted setting. We intend to generalize observational consistency of [9], and to understand its semantics building on the results of the present paper; ultimately employing it in a larger study of decomposition.

References

1. Larsen, K.G., Thomsen, B.: A modal process logic. In: LICS, IEEE Computer Society Press, Los Alamitos (1988)
2. Huth, M., Jagadeesan, R., Schmidt, D.: Modal transition systems: A foundation for three-valued program analysis. In: Sands, D. (ed.) ESOP 2001 and ETAPS 2001. LNCS, vol. 2028, Springer, Heidelberg (2001)
3. Schmidt, D.: From trace sets to modal-transition systems by stepwise abstract interpretation (2001)
4. Godefroid, P., Huth, M., Jagadeesan, R.: Abstraction-based model checking using modal transition systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, p. 426. Springer, Heidelberg (2001)
5. Børjesson, A., Larsen, K.G., Skou, A.: Generality in design and compositional verification using tav. In: FORTE '92 Proceedings, The Netherlands, pp. 449–464. North-Holland Publishing Co., Amsterdam (1993)
6. Larsen, K.G., Steffen, B., Weise, C.: A constraint oriented proof methodology based on modal transition systems. In: Tools and Algorithms for Construction and Analysis of Systems, pp. 17–40 (1995)
7. Bruns, G.: An industrial application of modal process logic. Sci. Comput. Program. 29(1-2), 3–22 (1997)
8. Larsen, K.G., Xinxin, L.: Equation solving using modal transition systems. In: LICS. Fifth Annual IEEE Symposium on Logics in Computer Science, Philadelphia, PA, USA, 4–7 June 1990, pp. 108–117. IEEE Computer Society Press, Los Alamitos (1990)
9. Larsen, K.G., Nyman, U., Wąsowski, A.: Modal i/o automata for interface and product line theories. In: Nicola, R.D. (ed.) ESOP 2007. Programming Languages and Systems. LNCS, vol. 4421, pp. 64–79. Springer, Heidelberg (2007)

10. Fischbein, D., Uchitel, S., Braberman, V.: A foundation for behavioural conformance in software product line architectures. In: ROSATEA '06 Proceedings, pp. 39–48. ACM Press, New York, NY, USA (2006)
11. Uchitel, S., Chechik, M.: Merging partial behavioural models. In: Taylor, R.N., Dwyer, M.B. (eds.) SIGSOFT FSE, pp. 43–52. ACM Press, New York (2004)
12. Brunet, G., Chechik, M., Uchitel, S.: Properties of behavioural model merging. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 98–114. Springer, Heidelberg (2006)
13. Weise, C., Lenzkes, D.: Weak refinement for modal hybrid systems. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 316–330. Springer, Heidelberg (1997)
14. Larsen, K.G.: Modal specifications. In: Sifakis, J. (ed.) Automatic Verification Methods for Finite State Systems. LNCS, vol. 407, pp. 232–246. Springer, Heidelberg (1990)
15. Cerans, K., Godskesen, J.C., Larsen, K.G.: Timed modal specification - theory and tools. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 253–267. Springer, Heidelberg (1993)
16. Larsen, K.G., Steffen, B., Weise, C.: Fischer's protocol revisited: a simple proof using modal constraints. In: Alur, R., Sontag, E.D., Henzinger, T.A. (eds.) Hybrid Systems III. LNCS, vol. 1066, pp. 604–615. Springer, Heidelberg (1996)
17. Fecher, H., Huth, M.: Ranked predicate abstraction for branching time: Complete incremental, and precise. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 322–336. Springer, Heidelberg (2006)
18. Schmidt, H., Fecher, H.: Comparing disjunctive modal transition systems with a one-selecting variant (submitted for publication) (2007)
19. Hüttel, H., Larsen, K.G.: The use of static constructs in a modal process logic. In: LFCS. The 1st International Symposium on Logical Foundations of Computer Science (1989)
20. Dams, D.: Abstract Interpretation and Partition Refinement for Model Checking. PhD thesis, Eindhoven University of Technology (July 1996)
21. Henessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. Journal of the ACM, 137–161 (1985)
22. Larsen, K.G.: A context dependent bisimulation between processes. Theoretical Computer Science 49 (1987)
23. Park, D.: Concurrency and automata on infinite sequences. In: Proceedings of 5th GI Conference, vol. 104 (1981)
24. Milner, R.: Calculi for synchrony and asynchrony. Theoretical Computer Science 25 (1983)
25. Godefroid, P., Jagadeesan, R.: Automatic abstraction using generalized model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 137–150. Springer, Heidelberg (2002)
26. Hüttel, H.: Operational and denotational properties of modal process logic. Master's thesis, Computer Science Department. Aalborg University (1988)
27. Xinxin, L.: Specification and Decomposition in Concurrency. PhD thesis, Department of Mathematics and Comnputer Science, Aalborg University (April 1992)
28. Alur, R., Henzinger, T.A., Kupferman, O., Vardi, M.: Alternating refinement relations. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 163–178. Springer, Heidelberg (1998)
29. Alfaro, L., Henzinger, T.A.: Interface automata. In: FSE. Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering, Vienna, Austria, pp. 109–120 (september 2001)

Equivalence Checking for Infinite Systems Using Parameterized Boolean Equation Systems

Taolue Chen[1,⋆], Bas Ploeger[2,⋆⋆], Jaco van de Pol[1,2], and Tim A.C. Willemse[2,⋆⋆⋆]

[1] CWI, Department of Software Engineering,
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
[2] Eindhoven University of Technology, Design and Analysis of Systems Group,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract. In this paper, we provide a transformation from the branching bisimulation problem for infinite, concurrent, data-intensive systems in linear process format, into solving Parameterized Boolean Equation Systems. We prove correctness, and illustrate the approach with an unbounded queue example. We also provide some adaptations to obtain similar transformations for weak bisimulation and simulation equivalence.

1 Introduction

A standard approach for verifying the correctness of a computer system or a communication protocol is the *equivalence-based methodology*. This framework was introduced by Milner [23] and has been intensively explored in process algebra. One proceeds by establishing two descriptions (models) for one system: a *specification* and an *implementation*. The former describes the desired high-level behavior, while the latter provides lower-level details indicating how this behavior is to be achieved. Then an implementation is said to be correct, if it behaves "the same as" its specification. Similarly, one could check whether the implementation has "at most" the behavior allowed by the specification. Several *behavioral equivalences and preorders* have been introduced to relate specifications and implementations, supporting different notions of observability. These include strong, weak [24], and branching bisimulation [11,4].

Equivalence Checking for Finite Systems. Checking strong bisimulation of finite systems can be done very efficiently. The basic algorithm is the well-known *partition refinement* algorithm [26]. For weak bisimulation checking, one could compute the transitive closure of τ-transitions, and thus lift the algorithms for strong bisimulation to the weak one. This is viable but costly, since it might incur a quadratic blow-up w.r.t.

⋆ This author is partially supported by Dutch Bsik project BRICKS, 973 Program of China (2002CB312002), NSF of China (60233010, 60273034, 60403014), 863 Program of China (2005AA113160, 2004AA112090).

⋆⋆ This author is partially supported by the Netherlands Organisation for Scientific Research (NWO) under VoLTS grant number 612.065.410.

⋆⋆⋆ This author is partially supported by the Netherlands Organisation for Scientific Research (NWO) under BRICKS/FOCUS grant number 642.000.602.

the original LTSs. Instead, one could employ the more efficient solution by [15] for checking branching bisimulation, as branching and weak bisimulation often coincide.

Alternatively, one can transform several bisimulation relations into Boolean Equation Systems (BES). Various encodings have been proposed in the literature [2,8,22], leading to efficient tools. In [2] it is shown that the BESs obtained from equivalence relations have a special format; the encodings of [22] even yield alternation free BESs (cf. definition of alternation depth in [21]) for up to five different behavioral equivalences. Solving alternation free BESs can be done very efficiently. However, finiteness of the graphs is crucial for the encodings yielding alternation free BESs.

It is interesting to note that the μ-calculus model checking problem for finite systems can also be transformed to the problem of solving a BES [2,21]. Hence, a BES solver, e.g. [22], provides a uniform engine for verification by model checking and equivalence checking for finite systems.

Our Contribution. In this paper, we focus on equivalence checking for *infinite* systems. Generally for concurrent systems with data, the induced labeled transition system (LTS) is no longer *finite*, and the traditional algorithms fail for *infinite* transition graphs. The symbolic approach needed for infinite systems depends on the specification format. We use *Linear Process Equations* (LPEs), which originate from μCRL [14], a process algebra with abstract data types, and describe the system by a finite set of guarded, nondeterministic transitions. LPEs are Turing complete, and many formalisms can be compiled to LPEs without considerable blow-up. Therefore, our methods essentially also apply to LOTOS [5], timed automata [1], I/O-automata [20], finite control π-calculus [25], UNITY [6], etc.

The solution we propose in this paper is inspired by [12], where the question whether an LPE satisfies a *first-order* μ-calculus formula is transformed into a *Parameterized Boolean Equation System* (PBES). PBESs extend boolean equation systems with data parameters and quantifiers. Heuristics, techniques [17], and tool support [16] have been developed for solving PBESs. This is still subject to ongoing research. Also in [28] such equation systems are used for model checking systems with data and time. In general, solving PBESs cannot be completely automated.

We propose to check branching bisimilarity of infinite systems by solving recursive equations. In particular, we show how to generate a PBES from two LPEs. The resulting PBES has alternation depth two. We prove that the PBES has a positive solution if and only if the two (infinite) systems are branching bisimilar. Moreover, we illustrate the technique by an example on unbounded queues, and show similar transformations for Milner's weak bisimulation [24] and branching simulation equivalence [10].

There are good reasons to translate branching bisimulation for infinite systems to solving PBESs, even though both problems are undecidable. The main reason is that solving PBESs is a more fundamental problem, as it boils down to solving equations between predicates. The other reason is that model checking mu-calculus with data has already been mapped to PBESs. Hence all efforts in solving PBESs (like [17]) can now be freely applied to the bisimulation problem as well.

Related Work. We already mentioned related work on finite systems, especially [2,22]. There are several approaches on which we want to comment in more detail.

The cones and foci method [9] rephrases the question whether two LPEs are bisimilar in terms of proof obligations on data objects. Basically, the user must first identify invariants, a focus condition, and a state mapping. In contrast, generating a PBES requires no human ingenuity, although solving the PBES still may. Furthermore, our solution is considerably more general, because it lifts two severe limitations of the cones and foci method. The first limitation is that the cones and foci method only works in case the branching bisimulation is functional (this means that a state in the implementation can only be related to a unique state in the specification). Another severe limitation of the cones and foci method is that it cannot handle specifications with τ-transitions. In some protocols (e.g. the bounded retransmission protocol [13]) this condition is not met and thus the cones and foci method fails. In our example on unbounded queues, both systems perform τ steps, and their bisimulation is not functional.

Our work can be seen as the generalization of [19] to weak and branching equivalences. In [19], Lin proposes Symbolic Transition Graphs with Assignments (STGA) as a new model for message-passing processes. An algorithm is also presented which computes bisimulation formulae for finite state STGAs, in terms of the greatest solutions of a *predicate equation system*. This corresponds to an alternation free PBES, and thus it can only deal with strong bisimulation.

The extension of Lin's work for strong bisimulation to weak and branching equivalences is not straightforward. This is testified by the encoding of weak bisimulation in predicate systems by Kwak *et al.* [18]. However, their encoding is not generally correct for STGA, as they use a conjunction over the complete τ-closure of a state. This only works in case that the τ-closure of every state is finite, which is generally not the case for STGA, also not for our LPEs. Alternation depth 2 seems unavoidable but does not occur in [18]. Note that for finite LTS a conjunction over the τ-closure is possible [22], but leads to a quadratic blow-up of the BES in the worst case.

Structure of the Paper. The paper is organized as follows. In Section 2, we provide background knowledge on linear process equations, labeled transition systems and bisimulation equivalences. We assume familiarity with standard fixpoint theory. In Section 3, PBESs are reviewed. Section 4 is devoted to the presentation of the translation and the justification of its correctness. In Section 5, we provide an example to illustrate the use of our algorithm. In Section 6, we demonstrate how to adapt the translation for branching bisimulation to weak bisimulations and simulation equivalence. The translation for strong bisimulation and an additional example are presented in [7]. The paper is concluded in Section 7.

2 Preliminaries

Linear process equations have been proposed as a *symbolic* representation of general (infinite) labeled transition systems. In an LPE, the behavior of a process is denoted as a state vector of typed variables, accompanied by a set of condition-action-effect rules. LPEs are widely used in µCRL [14], a language for specifying concurrent systems and protocols in an algebraic style. We mention that µCRL has complete automatic tool support to generate LPEs from µCRL specifications.

Definition 1 (Linear Process Equation). *A* linear process equation *is a parameterized equation taking the form*

$$M(d:D) = \sum_{a \in Act} \sum_{e_a : E_a} h_a(d, e_a) \implies a(f_a(d, e_a)) \cdot M(g_a(d, e_a))$$

where $f_a : D \times E_a \to D_a$*,* $g_a : D \times E_a \to D$ *and* $h_a : D \times E_a \to \mathbb{B}$ *for each* $a \in Act$*. Note that here* D*,* D_a *and* E_a *are general data types and* $\mathbb{B}$ *is the boolean type.*

In the above definition, the LPE M specifies that if in the current state d the condition $h_a(d, e_a)$ holds for any e_a of sort E_a, then an action a carrying data parameter $f_a(d, e_a)$ is possible and the effect of executing this action is the new state $g_a(d, e_a)$. The values of the condition, action parameter and new state may depend on the current state and a summation variable e_a.

For simplicity and without loss of generality, we restrict ourselves to a single variable at the left-hand side in all our theoretical considerations and to the use of non-terminating processes. That is, we do not consider processes that, apart from executing an infinite number of actions, also have the possibility to perform a finite number of actions and then terminate successfully. Including multiple variables and termination in our theory does not pose any theoretical challenges, but is omitted from our exposition for brevity. The operational semantics of LPEs is defined in terms of *labeled transition systems*.

Definition 2 (Labeled Transition System). *The* labeled transition system *of an LPE (as defined in Definition 1) is a quadruple* $\mathcal{M} = \langle \mathcal{S}, \Sigma, \to, s_0 \rangle$*, where*

- $\mathcal{S} = \{d \mid d \in D\}$ *is the (possibly infinite) set of states;*
- $\Sigma = \{a(d) \mid a \in Act \wedge d \in D_a\}$ *is the (possibly infinite) set of labels;*
- $\to = \{(d, a(d'), d'') \mid a \in Act \wedge \exists e_a \in E_a . h_a(d, e_a) \wedge d' = f_a(d, e_a) \wedge d'' = g_a(d, e_a)\}$ *is the transition relation;*
- $s_0 = d_0 \in \mathcal{S}$*, for a given* $d_0 \in D$*, is the initial state.*

For an LPE M, we usually write $d \xrightarrow{a(d')}_M d''$ to denote the fact that $(d, a(d'), d'')$ is in the transition relation of the LTS of M. We will omit the subscript M when it is clear from the context. Following Milner [24], the derived transition relation $\Rightarrow$ is defined as the reflexive, transitive closure of $\xrightarrow{\tau}$ (i.e. $(\xrightarrow{\tau})^*$), and $\overset{\alpha}{\Rightarrow}$, $\overset{\hat{\alpha}}{\Rightarrow}$ and $\xrightarrow{\bar{\alpha}}$ are defined in the standard way as follows:

$$\overset{\alpha}{\Rightarrow} \overset{\text{def}}{=} \Rightarrow \xrightarrow{\alpha} \Rightarrow \qquad \overset{\hat{\alpha}}{\Rightarrow} \overset{\text{def}}{=} \begin{cases} \Rightarrow \text{ if } \alpha = \tau \\ \overset{\alpha}{\Rightarrow} \text{ otherwise.} \end{cases} \qquad \xrightarrow{\bar{\alpha}} \overset{\text{def}}{=} \begin{cases} \xrightarrow{\tau} \cup \text{Id if } \alpha = \tau \\ \xrightarrow{\alpha} \quad \text{otherwise.} \end{cases}$$

2.1 Bisimulation Equivalences

We now introduce several well-known equivalences. The definitions below are with respect to an arbitrary, given labeled transition system $\mathcal{M} = \langle S, \Sigma, \to, s_0 \rangle$.

Definition 3 (Branching (Bi)simulations). *A binary relation* $\mathcal{R} \subseteq S \times S$ *is a* semi-branching simulation*, iff whenever* $s\mathcal{R}t$ *then for all* $\alpha \in \Sigma$ *and* $s' \in S$*, if* $s \xrightarrow{\alpha} s'$*, then* $t \Rightarrow t' \xrightarrow{\bar{\alpha}} t''$ *for some* $t', t'' \in S$ *such that* $s\mathcal{R}t'$ *and* $s'\mathcal{R}t''$*. We say that:*

– $\mathcal{R}$ *is a* semi-branching bisimulation*, if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are semi-branching simulations.*
– s *is* branching bisimilar *to t, denoted by $s \leftrightarroweq_b t$, iff there exists a semi-branching bisimulation $\mathcal{R}$, such that $s\mathcal{R}t$.*
– s *is* branching simulation equivalent *to t, iff there exist $\mathcal{R}$ and $\mathcal{Q}$, such that $s\mathcal{R}t$ and $t\mathcal{Q}s$ and both $\mathcal{R}$ and $\mathcal{Q}$ are semi-branching simulations.*

Note that although a semi-branching simulation is not necessarily a branching simulation, it is shown in [4] that this definition of branching bisimilarity coincides with the original definition in [11]. Therefore, in the sequel we take the liberty to use *semi-branching* and *branching* interchangeably. In the theoretical considerations in this paper, semi-branching relations are more convenient as they allow for shorter and clearer proofs of our theorems.

Definition 4 (Weak Bisimulation). *A binary relation $\mathcal{R} \subseteq S \times S$ is an (early)* weak bisimulation*, iff it is symmetric and whenever $s\mathcal{R}t$ then for all $\alpha \in \Sigma$ and $s' \in S$, if $s \xrightarrow{\alpha} s'$, then $t \overset{\hat{\alpha}}{\Rightarrow} t'$ for some $t' \in S$ such that $s'\mathcal{R}t'$.*
Weak bisimilarity*, denoted by $\leftrightarroweq_w$, is the largest weak bisimulation.*

3 Parameterized Boolean Equation Systems

A Parameterized Boolean Equation System (PBES) is a sequence of equations of the form

$$\sigma X(d : D) = \phi$$

σ denotes either the minimal (μ) or the maximal (ν) fixpoint. X is a predicate variable (from a set $\mathcal{P}$ of predicate variables) that binds a data variable d (from a set $\mathcal{D}$ of data variables) that may occur freely in the *predicate formula* ϕ. Apart from data variable d, ϕ can contain data terms, boolean connectives, quantifiers over (possibly infinite) data domains, and predicate variables. Predicate formulae ϕ are formally defined as follows:

Definition 5 (Predicate Formula). *A* predicate formula *is a formula ϕ in positive form, defined by the following grammar:*

$$\phi ::= b \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \forall d : D.\phi \mid \exists d : D.\phi \mid X(e)$$

where b is a data term of sort $\mathbb{B}$, possibly containing data variables $d \in \mathcal{D}$. Furthermore, $X \in \mathcal{P}$ is a (parameterized) predicate variable and e is a data term.

Note that negation does not occur in predicate formulae, except as an operator in data terms. We use $b \implies \phi$ as a shorthand for $\neg b \vee \phi$ for terms b of sort $\mathbb{B}$.

The semantics of predicates is dependent on the semantics of data terms. For a closed term e, we assume an interpretation function $[\![e]\!]$ that maps e to the data element it represents. For open terms, we use a *data environment* ε that maps each variable from $\mathcal{D}$ to a data value of the right sort. The interpretation of an open term e is denoted as $[\![e]\!]\varepsilon$ in the standard way.

Definition 6 (Semantics). *Let* $\theta : \mathcal{P} \to \wp(D)$ *be a* predicate environment *and* $\varepsilon : \mathcal{D} \to D$ *be a* data environment. *The interpretation of a predicate formula* ϕ *in the context of environment* θ *and* ε*, written as* $[\![\phi]\!]\theta\varepsilon$*, is either true or false, determined by the following induction:*

$$\begin{aligned}
[\![b]\!]\theta\varepsilon &= [\![b]\!]\varepsilon \\
[\![\phi_1 \wedge \phi_2]\!]\theta\varepsilon &= [\![\phi_1]\!]\theta\varepsilon \textit{ and } [\![\phi_2]\!]\theta\varepsilon \\
[\![\phi_1 \vee \phi_2]\!]\theta\varepsilon &= [\![\phi_1]\!]\theta\varepsilon \textit{ or } [\![\phi_2]\!]\theta\varepsilon \\
[\![\forall d : D.\phi]\!]\theta\varepsilon &= \textit{for all } v \in D,\ [\![\phi]\!]\theta(\varepsilon[v/d]) \\
[\![\exists d : D.\phi]\!]\theta\varepsilon &= \textit{there exists } v \in D,\ [\![\phi]\!]\theta(\varepsilon[v/d]) \\
[\![X(e)]\!]\theta\varepsilon &= \textit{true if } [\![e]\!]\varepsilon \in \theta(X) \textit{ and false otherwise}
\end{aligned}$$

Definition 7 (Parameterized Boolean Equation System). *A* parameterized boolean equation system *is a finite sequence of equations of the form* $\sigma X(d : D) = \phi$ *where* ϕ *is a predicate formula in which at most* d *may occur as a free data variable. The empty equation system is denoted by* ϵ*.*

In the remainder of this paper, we abbreviate parameterized boolean equation system to *equation system*. We say an equation system is *closed* whenever every predicate variable occurring at the right-hand side of some equation occurs at the left-hand side of some equation. The *solution* to an equation system is defined in the context of a predicate environment, as follows.

Definition 8 (Solution to an Equation System). *Given a predicate environment* θ *and an equation system* $\mathcal{E}$*, the* solution $[\![\mathcal{E}]\!]\theta$ *to* $\mathcal{E}$ *is an environment that is defined as follows, where* σ *is the greatest or least fixpoint, defined over the complete lattice* $\wp(D)$*.*

$$\begin{aligned}
[\![\epsilon]\!]\theta &= \theta \\
[\![(\sigma X(d : D) = \phi)\mathcal{E}]\!]\theta &= [\![\mathcal{E}]\!](\theta\Big[\sigma\mathcal{X}{\in}\wp(D).\lambda v{\in}D.[\![\phi]\!]([\![\mathcal{E}]\!]\theta[\mathcal{X}/X])[v/d]/X\Big])
\end{aligned}$$

For *closed* equation systems, the solution for the binding predicate variables does not depend on the given environment θ. In such cases, we refrain from writing the environment explicitly.

4 Translation for Branching Bisimulation

We define a translation that encodes the problem of finding the largest branching bisimulation in the problem of solving an equation system.

Definition 9. *Let* M *and* S *be LPEs of the following form:*

$$M(d : D^{\mathsf{M}}) = \sum_{a \in Act} \sum_{e_a : E_a^{\mathsf{M}}} h_a^{\mathsf{M}}(d, e_a) \implies a(f_a^{\mathsf{M}}(d, e_a)).M(g_a^{\mathsf{M}}(d, e_a))$$

$$S(d : D^{\mathsf{S}}) = \sum_{a \in Act} \sum_{e_a : E_a^{\mathsf{S}}} h_a^{\mathsf{S}}(d, e_a) \implies a(f_a^{\mathsf{S}}(d, e_a)).S(g_a^{\mathsf{S}}(d, e_a))$$

Given initial states $d : D^{\mathsf{M}}$ *and* $d' : D^{\mathsf{S}}$*, the equation system that corresponds to the branching bisimulation between LPEs* $M(d)$ *and* $S(d')$ *is constructed by the function* $brbisim$ *(see Algorithm 1).*

The main function $brbisim$ returns an equation system in the form $\nu E_2 \mu E_1$ where the bound predicate variables in E_2 are denoted by X and that in E_1 are denoted by Y. Intuitively, E_2 is used to characterize the (branching) bisimulation while E_1 is used to absorb the τ actions. The equation system's predicate formulae are constructed from the syntactic ingredients from LPEs M and S. Note that although we talk about the model (M) and the specification (S), the two systems are treated completely symmetrically. As we will show in Theorem 2, the solution for $X^{\mathsf{M},\mathsf{S}}$ in the resulting equation system gives the largest branching bisimulation relation between M and S as a predicate on $D^{\mathsf{M}} \times D^{\mathsf{S}}$.

Algorithm 1. Generation of a PBES for Branching Bisimulation

```
brbisim= $\nu E_2 \mu E_1$, where
        $E_2 := \{X^{\mathsf{M},\mathsf{S}}(d : D^{\mathsf{M}}, d' : D^{\mathsf{S}}) = match^{\mathsf{M},\mathsf{S}}(d, d') \wedge match^{\mathsf{S},\mathsf{M}}(d', d)$ ,
                $X^{\mathsf{S},\mathsf{M}}(d' : D^{\mathsf{S}}, d : D^{\mathsf{M}}) = X^{\mathsf{M},\mathsf{S}}(d, d')$ }
        $E_1 := \{Y_a^{\mathsf{p},\mathsf{q}}(d : D^{\mathsf{p}}, d' : D^{\mathsf{q}}, e : E_a^{\mathsf{p}}) = close_a^{\mathsf{p},\mathsf{q}}(d, d', e)$
                $\mid a \in Act \wedge (\mathsf{p}, \mathsf{q}) \in \{(\mathsf{M}, \mathsf{S}), (\mathsf{S}, \mathsf{M})\}\}$

Where we use the following abbreviations, for all $a \in Act \wedge (p, q) \in \{(M, S), (S, M)\}$:

$match^{\mathsf{p},\mathsf{q}}(d : D^{\mathsf{p}}, d' : D^{\mathsf{q}}) = \bigwedge_{a \in Act} \forall e : E_a^{\mathsf{p}}. (h_a^{\mathsf{p}}(d, e) \implies Y_a^{\mathsf{p},\mathsf{q}}(d, d', e))$;

$close_a^{\mathsf{p},\mathsf{q}}(d : D^{\mathsf{p}}, d' : D^{\mathsf{q}}, e : E_a^{\mathsf{p}}) = \exists e' : E_\tau^{\mathsf{q}}. (h_\tau^{\mathsf{q}}(d', e') \wedge Y_a^{\mathsf{p},\mathsf{q}}(d, g_\tau^{\mathsf{q}}(d', e'), e))$
        $\vee (X^{\mathsf{p},\mathsf{q}}(d, d') \wedge step_a^{\mathsf{p},\mathsf{q}}(d, d', e))$;

$step_a^{\mathsf{p},\mathsf{q}}(d : D^{\mathsf{p}}, d' : D^{\mathsf{q}}, e : E_a^{\mathsf{p}}) = (a = \tau \wedge X^{\mathsf{p},\mathsf{q}}(g_\tau^{\mathsf{p}}(d, e), d')) \vee$
        $\exists e' : E_a^{\mathsf{q}}. h_a^{\mathsf{q}}(d', e') \wedge (f_a^{\mathsf{p}}(d, e) = f_a^{\mathsf{q}}(d', e')) \wedge X^{\mathsf{p},\mathsf{q}}(g_a^{\mathsf{p}}(d, e), g_a^{\mathsf{q}}(d', e'))$;
```

4.1 Correctness of Transformation

In this section we confirm the relation between the branching bisimulation problem and the problem of solving an equation system. Before establishing the correctness of the transformation presented above, we first provide a fixpoint characterization for (semi-) branching bisimilarity, which we exploit in the correctness proof of our algorithm. For brevity, given any LPEs M and S, and any binary relation $\mathcal{B}$ over $D^{\mathsf{M}} \times D^{\mathsf{S}}$, we define a functional $\mathcal{F}$ as

$$
\begin{aligned}
&\mathcal{F}(\mathcal{B}) = \{(d, d') \mid \forall a \in Act, e_a \in E_a^{\mathsf{M}}.h_a^{\mathsf{M}}(d, e_a) \implies \\
&\quad \exists d_2', d_3'.d' \Rightarrow_S d_2' \wedge d_2' \xrightarrow{\overline{a(f_a^{\mathsf{M}}(d,e_a))}}_S d_3' \wedge (d, d_2') \in \mathcal{B} \wedge (g_a^{\mathsf{M}}(d, e_a), d_3') \in \mathcal{B}, \\
&\qquad \text{and } \forall a \in Act, e_a' \in E_a^{\mathsf{S}}.h_a^{\mathsf{S}}(d', e_a') \implies \\
&\quad \exists d_2, d_3.d \Rightarrow_M d_2 \wedge d_2 \xrightarrow{\overline{a(f_a^{\mathsf{S}}(d',e_a'))}}_M d_3 \wedge (d_2, d') \in \mathcal{B} \wedge (d_3, g_a^{\mathsf{S}}(d', e_a')) \in \mathcal{B}\}
\end{aligned}
$$

It is not difficult to see that $\mathcal{F}$ is monotonic. We claim that branching bisimilarity is the *maximal* fixpoint of functional $\mathcal{F}$ (i.e. $\nu B.\mathcal{F}(B)$).

Lemma 1. $\underline{\leftrightarrow}_b = \nu B.\mathcal{F}(B)$.

Proof. We prove set inclusion both ways using the definition of $\mathcal{F}$ and fixpoint theorems. The full proof is included in [7]. □

For proving the correctness of our translation, we first solve μE_1 given an arbitrary solution for X.

Theorem 1. *For any LPEs M and S, let μE_1 be generated by Algorithm 1, let η be an arbitrary predicate environment, and let $\theta = [\![\mu E_1]\!]\eta$. Then for any action a, and any d, d' and e, we have $(d, d', e) \in \theta(Y_a^{\mathsf{M,S}})$ if and only if*

$$\exists d_2, d_3.\, d' \Rightarrow_S d_2 \wedge d_2 \xrightarrow{\overline{a(f_a^{\mathsf{M}}(d,e))}}_S d_3 \wedge (d, d_2) \in \eta(X^{\mathsf{M,S}}) \wedge (g_a^{\mathsf{M}}(d,e), d_3) \in \eta(X^{\mathsf{M,S}})$$

Proof. We drop the superscripts $\mathsf{M,S}$ when no confusion arises. We define sets $\mathcal{R}_i^{a,d,e} \subseteq D^{\mathsf{S}}$, for any $a \in Act$, d, e, $i \geq 0$, and depending on $\eta(X)$, as follows:

$$\begin{cases} \mathcal{R}_0^{a,d,e} = \{d' \mid \exists d_3.\, d' \xrightarrow{\overline{a(f_a^{\mathsf{M}}(d,e))}}_S d_3 \wedge (d, d') \in \eta(X) \wedge (g_a^{\mathsf{M}}(d,e), d_3) \in \eta(X)\} \\ \mathcal{R}_{i+1}^{a,d,e} = \{d' \mid \exists d_2.\, d' \xrightarrow{\tau}_S d_2 \wedge d_2 \in R_i^{a,d,e}\} \end{cases}$$

And let $\mathcal{R}^{a,d,e} = \bigcup_{i \geq 0} \mathcal{R}_i^{a,d,e}$. Obviously, by definition of $\Rightarrow$, we have

$$\begin{aligned} \mathcal{R}^{a,d,e} = \{d' \mid \exists d_2, d_3.\, d' \Rightarrow_S d_2 \wedge d_2 \xrightarrow{\overline{a(f_a^{\mathsf{M}}(d,e))}}_S d_3 \wedge (d, d_2) \in \eta(X) \\ \wedge (g_a^{\mathsf{M}}(d,e), d_3) \in \eta(X)\} \end{aligned}$$

We will prove, using an approximation method, that this coincides with the minimal solution of $Y_a^{\mathsf{M,S}}$. More precisely, we claim:

$$((d, d', e) \in \theta(Y_a^{\mathsf{M,S}})) = (d' \in \mathcal{R}^{a,d,e})$$

Recall that according to the algorithm, Y_a is of the form

$$Y_a(d, d', e) = (X(d, d') \wedge \Xi) \vee \exists e'_\tau.(h_\tau^{\mathsf{S}}(d', e'_\tau) \wedge Y_a(d, g_\tau^{\mathsf{S}}(d', e'_\tau), e)) \tag{1}$$

where Ξ (generated by function *step*) is of the form

$$\begin{aligned} &(a = \tau \wedge X(g_\tau^{\mathsf{M}}(d,e), d')) \vee \\ &\ \exists e'_a.h_a^{\mathsf{S}}(d', e'_a) \wedge (f_a^{\mathsf{M}}(d,e) = f_a^{\mathsf{S}}(d', e'_a)) \wedge X(g_a^{\mathsf{M}}(d,e), g_a^{\mathsf{S}}(d', e'_a)) \end{aligned}$$

Note that, using the operational semantics for LPE S,

$$[\![X(d, d') \wedge \Xi]\!]\eta = \exists d''.\, (d, d') \in \eta(X) \wedge (g_a^{\mathsf{M}}(d,e), d'') \in \eta(X) \wedge d' \xrightarrow{\overline{a(f_a^{\mathsf{M}}(d,e))}}_S d''$$

Hence,

$$[\![X(d, d') \wedge \Xi]\!]\eta = (d' \in \mathcal{R}_0^{a,d,e}) \tag{2}$$

We next show by induction on n, that the finite approximations $Y_a^n(d,d',e)$ of equation (1) can be characterized by the following equation:

$$Y_a^n(d,d',e) = (d' \in \bigcup_{0 \le i < n} \mathcal{R}_i^{a,d,e})$$

The basis is trivial ($Y_a = \emptyset$). For the induction step, it suffices to note that

$$\begin{aligned}
&\{d' \mid Y_a^{n+1}(d,d',e)\} \\
&\overset{*}{=} \{d' \mid ((d,d') {\in} \eta(X) \wedge [\![\varXi]\!]\eta) \vee \exists e'_\tau.(h_\tau^{\mathsf{S}}(d',e'_\tau) \wedge g_\tau^{\mathsf{S}}(d',e'_\tau) \in \bigcup_{0 \le i < n} \mathcal{R}_i^{a,d,e})\} \\
&= \{d' \mid (d,d') {\in} \eta(X) \wedge [\![\varXi]\!]\eta\} \cup \bigcup_{0 \le i < n} \{d' \mid \exists e'_\tau.(h_\tau^{\mathsf{S}}(d',e'_\tau) \wedge g_\tau^{\mathsf{S}}(d',e'_\tau) \in \mathcal{R}_i^{a,d,e})\} \\
&\overset{\bigstar}{=} \mathcal{R}_0^{a,d,e} \cup \bigcup_{0 \le i < n} \mathcal{R}_{i+1}^{a,d,e} \\
&= \bigcup_{0 \le i < n+1} \mathcal{R}_i^{a,d,e},
\end{aligned}$$

where the step ($*$) uses the induction hypothesis, and the step ($\bigstar$) uses equation (2) above, and the definition of $\mathcal{R}_i^{a,d,e}$.

Next we compute the first infinitary approximation Y_a^ω of equation (1):

$$\begin{aligned}
\{d' \mid Y_a^\omega(d,d',e)\} &= \bigcup_{n \ge 0} \{d' \mid Y_a^n(d,d',e)\} \\
&= \bigcup_{n \ge 0} \bigcup_{0 \le i < n} \mathcal{R}_i^{a,d,e} \\
&= \bigcup_{i \ge 0} \mathcal{R}_i^{a,d,e}
\end{aligned}$$

It remains to show that the solution is stable, i.e. Y^ω is a solution of equation (1). This can be readily checked as follows:

$$\begin{aligned}
&\{d' \mid ((d,d') \in \eta(X) \wedge [\![\varXi]\!]\eta) \vee \exists e'_\tau.(h_\tau^{\mathsf{S}}(d',e'_\tau) \wedge g_\tau^{\mathsf{M}}(d',e'_\tau) \in \mathcal{R}^{a,d,e})\} \\
&= \mathcal{R}_0 \cup \bigcup_{i \ge 1} \mathcal{R}_i^{a,d,e} \\
&= \mathcal{R}^{a,d,e}
\end{aligned}$$

Hence we have found the correct minimal solution of μE_1. □

Finally, the correctness of the algorithm follows from the following theorem.

Theorem 2. *Let $\nu E_2 \mu E_1$ be the equation system generated by Algorithm 1 on M and S and $\theta = [\![\nu E_2 \mu E_1]\!]$. Then for all d and d' we have $M(d) \underline{\leftrightarrow}_b S(d')$ if and only if $(d,d') \in \theta(X^{\mathsf{M},\mathsf{S}})$.*

Proof. Recall that according to the algorithm, $X^{\mathsf{M},\mathsf{S}}$ is of the form

$$X^{\mathsf{M},\mathsf{S}}(d,d') = \bigwedge_{a\in Act} \forall e_a.(h_a^{\mathsf{M}}(d,e_a) \implies Y_a^{\mathsf{M},\mathsf{S}}(d,d',e_a)) \\ \wedge \bigwedge_{a\in Act} \forall e'_a.(h_a^{\mathsf{S}}(d',e'_a) \implies Y_a^{\mathsf{S},\mathsf{M}}(d',d,e'_a))$$

By symmetry, w.l.o.g. we only consider $\bigwedge_{a\in Act} \forall e_a.(h_a^{\mathsf{M}}(d,e_a) \implies Y_a^{\mathsf{M},\mathsf{S}}(d,d',e_a))$.
We define $G : D^{\mathsf{M}} \times D^{\mathsf{S}} \to D^{\mathsf{M}} \times D^{\mathsf{S}}$ as

$$G(\mathcal{B}) = \{(d,d') \mid \bigwedge_{a\in Act} \forall e_a.(h_a^{\mathsf{M}}(d,e_a) \implies (d,d',e_a) \in \eta(Y_a^{\mathsf{M},\mathsf{S}}))\}$$

where $\eta = [\![\mu E_1]\!][\mathcal{B}/X^{\mathsf{M},\mathsf{S}}]$.

Note that by [17, Lemma 5], G is monotonic, and thus the maximal fixpoint of G exists which is denoted by $\nu B.G(B)$. According to the semantics of PBES (cf. Definition 8), we have

$$\nu B.G(B) = \{(d,d') \mid (d,d') \in \theta(X^{\mathsf{M},\mathsf{S}})\}$$

Recall that the functional $\mathcal{F}$ is defined as

$$\mathcal{F}(\mathcal{B}) = \{(d,d') \mid \forall a \in Act, e_a \in E_a.h_a^{\mathsf{M}}(d,e_a) \implies \\ \exists d_2,d_3.d' \Rightarrow d_2 \wedge d_2 \xrightarrow{\overline{a(f_a^{\mathsf{M}}(d,e_a))}}_S d_3 \wedge (d,d_2) \in \mathcal{B} \wedge (g_a^{\mathsf{M}}(d,e_a),d_3) \in \mathcal{B}\}$$

We claim that for any $\mathcal{B}$,

$$\mathcal{F}(\mathcal{B}) = G(\mathcal{B})$$

To see this, first let us note that by Theorem 1

$$\eta(Y_a^{\mathsf{M},\mathsf{S}}) = \{d' \mid \exists d_2,d_3.\, d' \Rightarrow d_2 \wedge d_2 \xrightarrow{\overline{a(f_a^{\mathsf{M}}(d,e))}}_S d_3 \wedge \mathcal{B}(d,d_2) \wedge \mathcal{B}(g_a^{\mathsf{M}}(d,e),d_3)\}$$

It follows that

$$\begin{aligned} & G(\mathcal{B}) \\ &= \{(d,d') \mid \bigwedge_{a\in Act} \forall e_a.(h_a(d,e_a) \implies (d,d',e_a) \in \eta(Y_a^{\mathsf{M},\mathsf{S}})\} \\ &= \{(d,d') \mid \bigwedge_{a\in Act} \forall e_a.(h_a(d,e_a) \implies \exists d_2,d_3.\, d' \Rightarrow_S d_2 \wedge d_2 \xrightarrow{\overline{a(f_a^{\mathsf{M}}(d,e))}}_S d_3 \wedge \\ &\qquad \mathcal{B}(d,d_2) \wedge \mathcal{B}(g_a^{\mathsf{M}}(d,e),d_3)\} \\ &= \mathcal{F}(\mathcal{B}) \end{aligned}$$

It follows from Lemma 1 that

$$\underline{\leftrightarrow}_b = \nu\mathcal{F} = \nu B.G(B) = \{(d,d') \mid (d,d') \in \theta(X^{\mathsf{M},\mathsf{S}})\}$$

from which it is not difficult to see that $(d,d') \in \theta(X)$ if and only if $M(d) \underline{\leftrightarrow}_b S(d')$.
□

5 Example: Unbounded Queues

In this section we demonstrate the potential of the technique outlined in the previous section by applying it to an example of unbounded queues. The capacity of a bounded queue is doubled by connecting a queue of the same capacity. This means that a composition of bounded queues is behaviorally different from the constituent queues. In contrast, a composition of queues with infinite capacity does not change the behavior, as this again yields an unbounded queue.

Let D be an arbitrary data sort (possibly infinite sized) which is equipped with an equality relation, and let $\mathcal{Q}$ denote the data sort of queues of infinite capacity. We denote the empty queue by $[]$ and for any $d \in D$ we denote the queue containing only d by $[d]$. Operations on queues include $q +\!\!+ q'$, denoting the natural concatenation of queues q and q', and functions $hd : \mathcal{Q} \to D$ and $tl : \mathcal{Q} \to \mathcal{Q}$ which yield the head and tail of a queue q, respectively.

The processes S and T defined below model the composition of two unbounded queues and three unbounded queues, respectively. Remark that we obtained LPEs S and T as a result of an automated linearization of the parallel composition of two (resp. three) queues of infinite capacity. These original specifications have been omitted for brevity. Processes S and T can communicate with their environments via parameterized actions $r(d)$ (read d from the environment) and $w(d)$ (write d to the environment). The τ actions represent the internal communication of data from one queue to the next.

$$\begin{array}{l}
S(s_0, s_1 : \mathcal{Q}) = \\
\quad \sum_{v:D} r(v) \cdot S([v] +\!\!+ s_0, s_1) \\
+ s_1 \neq [] \implies w(hd(s_1)) \cdot S(s_0, tl(s_1)) \\
+ s_0 \neq [] \implies \tau \cdot S(tl(s_0), [hd(s_0)] +\!\!+ s_1)
\end{array}
\qquad
\begin{array}{l}
T(t_0, t_1, t_2 : \mathcal{Q}) = \\
\quad \sum_{u:D} r(u) \cdot T([u] +\!\!+ t_0, t_1, t_2) \\
+ t_2 \neq [] \implies w(hd(t_2)) \cdot T(t_0, t_1, tl(t_2)) \\
+ t_0 \neq [] \implies \tau \cdot T(tl(t_0), [hd(t_0)] +\!\!+ t_1, t_2) \\
+ t_1 \neq [] \implies \tau \cdot T(t_0, tl(t_1), [hd(t_1)] +\!\!+ t_2)
\end{array}$$

Applying Algorithm 1 for processes S and T, we obtain a PBES consisting of 8 equations. For lack of space, only the two most interesting fragments of the PBES are shown below.

$$\begin{array}{l}
\big(\nu X^{S,T}(s_0, s_1, t_0, t_1, t_2 : \mathcal{Q}) = \ldots \wedge \ (s_1 \neq [] \implies Y_w^{S,T}(s_0, s_1, t_0, t_1, t_2)) \ \wedge \ldots\big) \\
\vdots \\
\big(\mu Y_w^{S,T}(s_0, s_1, t_0, t_1, t_2 : \mathcal{Q}) = \ (t_0 \neq [] \wedge Y_w^{S,T}(s_0, s_1, tl(t_0), [hd(t_0)] +\!\!+ t_1, t_2)) \vee \\
\quad (t_1 \neq [] \wedge Y_w^{S,T}(s_0, s_1, t_0, tl(t_1), [hd(t_1)] +\!\!+ t_2)) \vee (t_2 \neq [] \wedge hd(t_2) = hd(s_1) \wedge \\
\quad X^{S,T}(s_0, s_1, t_0, t_1, t_2) \wedge X^{S,T}(s_0, tl(s_1), t_0, t_1, tl(t_2)))\ \big) \\
\vdots
\end{array}$$

In the remainder of this section, we strongly rely on techniques for solving and manipulating PBESs like adding invariants, symbolic approximations and strengthening equations. Some of these techniques have already been automated (e.g. symbolic approximation, see [16]). For a detailed account of all techniques, we refer to [16,17].

Consider the equation for $Y_w^{S,T}$. It represents the case where process T has to simulate a $w(hd(s_1))$ action of process S by possibly executing a finite number of τ-steps before executing action $w(hd(t_2))$. Inspired by the scenario that captures the minimal amount of τ-steps that are needed (two steps when $t_1 = t_2 = []$, one when $t_2 = [] \neq t_1$ and none otherwise), we strengthen the equation for $Y_w^{S,T}$ as follows:

$$\begin{aligned}
&\mu Y_w^{S,T}(s_0,s_1,t_0,t_1,t_2:\mathcal{Q}) = \\
&\quad (t_0 \neq [] \wedge \underline{t_1 = t_2 = []} \wedge Y_w^{S,T}(s_0,s_1,\mathit{tl}(t_0),[\mathit{hd}(t_0)]{+\!\!+}t_1,t_2)) \vee \\
&\quad (t_1 \neq [] \wedge \underline{t_2 = []} \wedge Y_w^{S,T}(s_0,s_1,t_0,\mathit{tl}(t_1),[\mathit{hd}(t_1)]{+\!\!+}t_2)) \vee \\
&\quad (t_2 \neq [] \wedge \mathit{hd}(t_2) = \mathit{hd}(s_1) \wedge X^{S,T}(s_0,s_1,t_0,t_1,t_2) \wedge X^{S,T}(s_0,\mathit{tl}(s_1),t_0,t_1,\mathit{tl}(t_2)))
\end{aligned}$$

The solution to $Y_w^{S,T}$ can be found by a straightforward symbolic approximation. This stabilizes at the fourth approximation, and can — depending on the rewriting technology that is used — be found automatically. The resulting solution is:

$$\begin{aligned}
&\mu Y_w^{S,T}(s_0,s_1,t_0,t_1,t_2:\mathcal{Q}) = \\
&\quad (t_0 \neq [] \wedge t_1 = t_2 = [] \wedge \mathit{hd}(t_0) = \mathit{hd}(s_1) \\
&\quad \wedge X^{S,T}(s_0,s_1,\mathit{tl}(t_0),[],[\mathit{hd}(t_0)]) \wedge X(s_0,\mathit{tl}(s_1),\mathit{tl}(t_0),[],[])) \vee \\
&\quad (t_1 \neq [] \wedge t_2 = [] \wedge \mathit{hd}(t_1) = \mathit{hd}(s_1) \wedge X^{S,T}(s_0,s_1,t_0,\mathit{tl}(t_1),[\mathit{hd}(t_1)]) \\
&\quad \wedge X^{S,T}(s_0,\mathit{tl}(s_1),t_0,\mathit{tl}(t_1),[])) \vee \\
&\quad (t_2 \neq [] \wedge \mathit{hd}(t_2) = \mathit{hd}(s_1) \wedge X^{S,T}(s_0,s_1,t_0,t_1,t_2) \wedge X^{S,T}(s_0,\mathit{tl}(s_1),t_0,t_1,\mathit{tl}(t_2)))
\end{aligned}$$

The solution to the (omitted) equation $Y_w^{T,S}$ can be obtained analogously. Likewise, we can strengthen and subsequently solve the equations for the Y_τ's and the Y_r's. The resulting solutions can be substituted in the equation for $X^{S,T}$ yielding the following closed equation for $X^{S,T}$.

$$\begin{aligned}
&\nu X^{S,T}(s_0,s_1,t_0,t_1,t_2:\mathcal{Q}) = \\
&\quad X^{S,T}(s_0,s_1,t_0,t_1,t_2) \wedge (\forall v:D\,.\,X^{S,T}([v]{+\!\!+}s_0,s_1,[v]{+\!\!+}t_0,t_1,t_2)) \\
&\wedge (s_1 \neq [] \implies ((t_0 \neq [] \wedge t_1 = [] \wedge t_2 = [] \wedge \mathit{hd}(t_0) = \mathit{hd}(s_1) \wedge \\
&\qquad\quad X^{S,T}(s_0,s_1,\mathit{tl}(t_0),[],[\mathit{hd}(t_0)]) \wedge X^{S,T}(s_0,\mathit{tl}(s_1),\mathit{tl}(t_0),[],[])) \\
&\qquad \vee (t_1 \neq [] \wedge t_2 = [] \wedge \mathit{hd}(t_1) = \mathit{hd}(s_1) \wedge \\
&\qquad\quad X^{S,T}(s_0,s_1,t_0,\mathit{tl}(t_1),[\mathit{hd}(t_1)]) \wedge X^{S,T}(s_0,\mathit{tl}(s_1),t_0,\mathit{tl}(t_1),[])) \\
&\qquad \vee (t_2 \neq [] \wedge \mathit{hd}(t_2) = \mathit{hd}(s_1) \wedge X^{S,T}(s_0,s_1,t_0,t_1,t_2) \wedge \\
&\qquad\quad X^{S,T}(s_0,\mathit{tl}(s_1),t_0,t_1,\mathit{tl}(t_2))))) \\
&\wedge (s_0 \neq [] \implies (X^{S,T}(s_0,s_1,t_0,t_1,t_2) \wedge (X^{S,T}(\mathit{tl}(s_0),[\mathit{hd}(s_0)]{+\!\!+}s_1,t_0,t_1,t_2) \\
&\qquad \vee (t_0 \neq [] \wedge X^{S,T}(\mathit{tl}(s_0),[\mathit{hd}(s_0)]{+\!\!+}s_1,\mathit{tl}(t_0),[\mathit{hd}(t_0)]{+\!\!+}t_1,t_2)) \\
&\qquad \vee (t_1 \neq [] \wedge X^{S,T}(\mathit{tl}(s_0),[\mathit{hd}(s_0)]{+\!\!+}s_1,t_0,\mathit{tl}(t_1),[\mathit{hd}(t_1)]{+\!\!+}t_2))))) \\
&\wedge (t_2 \neq [] \implies ((s_0 \neq [] \wedge s_1 = [] \wedge \mathit{hd}(s_0) = \mathit{hd}(t_2) \wedge \\
&\qquad\quad X^{S,T}(\mathit{tl}(s_0),[\mathit{hd}(s_0)],t_0,t_1,t_2) \wedge X^{S,T}(\mathit{tl}(s_0),[],t_0,t_1,\mathit{tl}(t_2))) \\
&\qquad \vee (s_1 \neq [] \wedge \mathit{hd}(s_1) = \mathit{hd}(t_2) \wedge X^{S,T}(s_0,s_1,t_0,t_1,t_2) \wedge \\
&\qquad\quad X^{S,T}(s_0,\mathit{tl}(s_1),t_0,t_1,\mathit{tl}(t_2))))) \\
\\
&\wedge ((t_0 \neq [] \vee t_1 \neq []) \implies (X^{S,T}(s_0,s_1,t_0,t_1,t_2) \wedge \\
&\qquad (X^{S,T}(s_0,s_1,\mathit{tl}(t_0),[\mathit{hd}(t_0)]{+\!\!+}t_1,t_2) \vee X^{S,T}(s_0,s_1,t_0,\mathit{tl}(t_1),[\mathit{hd}(t_1)]{+\!\!+}t_2) \vee \\
&\qquad X^{S,T}(\mathit{tl}(s_0),[\mathit{hd}(s_0)]{+\!\!+}s_1,t_0,\mathit{tl}(t_1),[\mathit{hd}(t_1)]{+\!\!+}t_2) \vee \\
&\qquad (s_0 \neq [] \wedge (X^{S,T}(\mathit{tl}(s_0),[\mathit{hd}(s_0)]{+\!\!+}s_1,\mathit{tl}(t_0),[\mathit{hd}(t_0)]{+\!\!+}t_1,t_2))))))
\end{aligned}$$

Utilizing the fact that $s_0 \mathbin{+\!\!+} s_1 = t_0 \mathbin{+\!\!+} t_1 \mathbin{+\!\!+} t_2$ is an invariant of the closed equation $X^{S,T}$, the symbolic approximation of $X^{S,T}$ stabilizes at the third approximation, yielding the solution $s_0 \mathbin{+\!\!+} s_1 = t_0 \mathbin{+\!\!+} t_1 \mathbin{+\!\!+} t_2$ [1]. Evaluating the solution to $X^{S,T}$ for the initial values $s_0 = s_1 = t_0 = t_1 = t_2 = []$ tells us that $S([], [])$ and $T([], [], [])$ are branching bisimilar. In fact, all processes $S(s_0, s_1)$ and $T(t_0, t_1, t_2)$ satisfying the condition $s_0 \mathbin{+\!\!+} s_1 = t_0 \mathbin{+\!\!+} t_1 \mathbin{+\!\!+} t_2$ are branching bisimilar.

6 Transformation for Other Equivalences

In this section, we demonstrate how we can adapt the algorithm presented in Section 4 to other variants of bisimulation. The strong case is simple and somehow known in [19] modulo different formalisms. The algorithm is included in [7]. As discussed in the introduction, our encoding for weak bisimulation (see Algorithm 2) fixes the generally incorrect encoding found in [18]. The case for (branching) simulation equivalence (see Algorithm 3) is novel. The correctness proofs are similar to the case for branching bisimulation.

Algorithm 2. Generation of a PBES for Weak Bisimulation

```
$\mathit{wbisim} = \nu E_2 \mu E_1$, where
    $E_2 := \{X^{\mathsf{M,S}}(d : D^{\mathsf{M}}, d' : D^{\mathsf{S}}) = \mathit{match}^{\mathsf{M,S}}(d, d') \wedge \mathit{match}^{\mathsf{S,M}}(d', d)$ ,
        $X^{\mathsf{S,M}}(d' : D^{\mathsf{S}}, d : D^{\mathsf{M}}) = X^{\mathsf{M,S}}(d, d')$ }
    $E_1 := \{Y^{\mathsf{p,q}}_{1,a}(d : D^{\mathsf{p}}, d' : D^{\mathsf{q}}, e : E^{\mathsf{p}}_a) = \mathit{close}^{\mathsf{p,q}}_{1,a}(d, d', e)$,
        $Y^{\mathsf{p,q}}_{2,a}(d : D^{\mathsf{p}}, d' : D^{\mathsf{q}}) = \mathit{close}^{\mathsf{p,q}}_{2,a}(d, d')$,
                $\mid a \in \mathit{Act} \wedge (\mathsf{p}, \mathsf{q}) \in \{(\mathsf{M}, \mathsf{S}), (\mathsf{S}, \mathsf{M})\}\}$

Where we use the following abbreviations, for all $a \in \mathit{Act} \wedge (\mathsf{p}, \mathsf{q}) \in \{(\mathsf{M}, \mathsf{S}), (\mathsf{S}, \mathsf{M})\}$:

$\mathit{match}^{\mathsf{p,q}}(d : D^{\mathsf{p}}, d' : D^{\mathsf{q}}) = \bigwedge_{a \in \mathit{Act}} \forall e : E^{\mathsf{p}}_a.(h^{\mathsf{p}}_a(d, e) \implies Y^{\mathsf{p,q}}_{1,a}(d, d', e))$;

$\mathit{close}^{\mathsf{p,q}}_{1,a}(d : D^{\mathsf{p}}, d' : D^{\mathsf{q}}, e : E^{\mathsf{p}}_a) = \exists e' : E^{\mathsf{q}}_{\tau}.(h^{\mathsf{q}}_{\tau}(d', e') \wedge Y^{\mathsf{p,q}}_{1,a}(d, g^{\mathsf{q}}_{\tau}(d', e'), e))$
        $\vee \mathit{step}^{\mathsf{p,q}}_a(d, d', e)$;

$\mathit{step}^{\mathsf{p,q}}_a(d : D^{\mathsf{p}}, d' : D^{\mathsf{q}}, e : E^{\mathsf{p}}_a) = (a = \tau \wedge \mathit{close}^{\mathsf{p,q}}_{2,a}(g^{\mathsf{p}}_a(d, e), d')) \vee$
    $\exists e' : E^{\mathsf{q}}_a.h^{\mathsf{q}}_a(d', e') \wedge (f^{\mathsf{p}}_a(d, e) = f^{\mathsf{q}}_a(d', e')) \wedge \mathit{close}^{\mathsf{p,q}}_{2,a}(g^{\mathsf{p}}_a(d, e), g^{\mathsf{q}}_a(d', e'))$ ;

$\mathit{close}^{\mathsf{p,q}}_{2,a}(d : D^{\mathsf{p}}, d' : D^{\mathsf{q}}) = X^{\mathsf{p,q}}(d, d') \vee \exists e' : E^{\mathsf{q}}_{\tau}.h^{\mathsf{q}}_{\tau}(d', e') \wedge Y^{\mathsf{p,q}}_{2,a}(d, g^{\mathsf{q}}_{\tau}(d', e'))$;
```

7 Conclusion

We have shown how to transform the weak and branching (bi)simulation equivalence checking problems for infinite systems to solving Parameterized Boolean Equation Systems. We demonstrated our method on a small example, showing that the concatenation of two unbounded queues is branching bisimilar to the concatenation of three

[1] Remark that the fact that the solution and the invariant match is coincidental: it is clear that e.g. the trivial invariant *true* ($\top$) does not exhibit this phenomenon.

Algorithm 3. Generation of a PBES for (Branching) Simulation Equivalence

```
brsim(m, n) = $\nu E_2 \mu E_1$, where
    $E_2 := \{X(d : D^{\mathsf{M}}, d' : D^{\mathsf{S}}) = X^{\mathsf{M,S}}(d, d') \wedge X^{\mathsf{S,M}}(d', d),$
          $X^{\mathsf{M,S}}(d : D^{\mathsf{M}}, d' : D^{\mathsf{S}}) = match^{\mathsf{M,S}}(d, d'),$
          $X^{\mathsf{S,M}}(d' : D^{\mathsf{S}}, d : D^{\mathsf{M}}) = match^{\mathsf{S,M}}(d', d)\}$
    $E_1 := \{Y_a^{\mathsf{p,q}}(m, n, e) = close_a^{\mathsf{p,q}}(d, d', e) \mid a \in Act\}$
```

Where we use the following abbreviations, for all $a \in Act \wedge (\mathsf{p}, \mathsf{q}) \in \{(\mathsf{M}, \mathsf{S}), (\mathsf{S}, \mathsf{M})\}$:

$$match^{\mathsf{p,q}}(d : D^{\mathsf{p}}, d' : D^{\mathsf{q}}) = \bigwedge_{a \in Act} \forall e : E_a^{\mathsf{p}}.\, (h_a^{\mathsf{p}}(d, e) \implies Y_a^{\mathsf{p,q}}(d, d', e));$$

$$close_a^{\mathsf{p,q}}(d : D^{\mathsf{p}}, d' : D^{\mathsf{q}}, e : E_a^{\mathsf{p}}) = \exists e' : E_\tau^{\mathsf{q}}.\, (h_\tau^{\mathsf{q}}(d', e') \wedge Y_a^{\mathsf{p,q}}(d, g_\tau^{\mathsf{q}}(d', e'), e)) \\ \vee (X^{\mathsf{p,q}}(d, d') \wedge step_a^{\mathsf{p,q}}(d, d', e));$$

$$step_a^{\mathsf{p,q}}(d : D^{\mathsf{p}}, d' : D^{\mathsf{q}}, e : E_a^{\mathsf{p}}) = (a = \tau \wedge X^{\mathsf{p,q}}(g_\tau^{\mathsf{p}}(d, e), d')) \vee \\ \exists e' : E_a^{\mathsf{q}}.\, h_a^{\mathsf{q}}(d', e') \wedge (f_a^{\mathsf{p}}(d, e) = f_a^{\mathsf{q}}(d', e')) \wedge X^{\mathsf{p,q}}(g_a^{\mathsf{p}}(d, e), g_a^{\mathsf{q}}(d', e'));$$

unbounded queues. This example could not be solved directly with the cones and foci method (without introducing a third process), because these systems are not functionally branching bisimilar, and moreover, both systems perform τ-steps.

Our solution is a symbolic verification algorithm. Compared with the previously known algorithms, it has the advantage that the solution of the PBES indicates exactly which states of the implementation and specification are bisimilar. This provides some positive feedback in case the initial states of the two systems are not bisimilar. Note that we have introduced a *generic* scheme that can be applied to other weak equivalences and preorders in branching time spectrum [10], and also to other formalisms of concurrency.

We conjecture that for infinite systems, it is essential that the PBES has alternation depth two, as opposed to the finite case. We leave it for future work to apply our method to various equivalences for mobile processes, in particular π-calculus [25], such as weak early, late and open bisimulation. Orthogonal to this, we shall continue our work on improving tool support for solving PBESs, and the application of our techniques to larger specifications of infinite systems.

Acknowledgments. We are grateful to Wan Fokkink and Jan Friso Groote for stimulating discussions.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Andersen, H.R.: Model checking and boolean graphs. Theoretical Computer Science 126(1), 3–30 (1994)
3. Andersen, H.R., Vergauwen, B.: Efficient checking of behavioural relations and modal assertions using fixed-point inversion. In: Wolper, P. (ed.) CAV 1995. LNCS, vol. 939, pp. 142–154. Springer, Heidelberg (1995)

4. Basten, T.: Branching bisimilarity is an equivalence indeed! Information Processing Letters 58, 141–147 (1996)
5. Bolognesi, T., Brinksma, E.: Introduction to the ISO specification language LOTOS. Computer Networks 14, 25–59 (1987)
6. Chandy, K.M., Misra, J.: Parallel Program Design: A Foundation. Addison-Wesley, Reading (1988)
7. Chen, T., Ploeger, B., van de Pol, J., Willemse, T.A.C.: Equivalence checking for infinite systems using parameterized boolean equation systems. CS-Report 07-14, Technische Universiteit Eindhoven (2007)
8. Cleaveland, R., Steffen, B.: Computing behavioural relations, logically. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) Automata, Languages and Programming. LNCS, vol. 510, pp. 127–138. Springer, Heidelberg (1991)
9. Fokkink, W., Pang, J., van de Pol, J.: Cones and foci: A mechanical framework for protocol verification. Formal Methods in System Design 29(1), 1–31 (2006)
10. van Glabbeek, R.: The Linear Time - Branching Time Spectrum II. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 66–81. Springer, Heidelberg (1993)
11. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. Journal of the ACM 43, 555–600 (1996)
12. Groote, J.F., Mateescu, R.: Verification of temporal properties of processes in a setting with data. In: Haeberer, A.M. (ed.) AMAST 1998. LNCS, vol. 1548, pp. 74–90. Springer, Heidelberg (1998)
13. Groote, J.F., van de Pol, J.: A bounded retransmission protocol for large data packets. In: Nivat, M., Wirsing, M. (eds.) AMAST 1996. LNCS, vol. 1101, pp. 536–550. Springer, Heidelberg (1996)
14. Groote, J.F., Reniers, M.: Algebraic process verification. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, pp. 1151–1208. Elsevier, Amsterdam (2001)
15. Groote, J.F., Vaandrager, F.W.: An efficient algorithm for branching bisimulation and stuttering equivalence. In: Paterson, M.S. (ed.) Automata, Languages and Programming. LNCS, vol. 443, pp. 626–638. Springer, Heidelberg (1990)
16. Groote, J.F., Willemse, T.A.C.: Model-checking processes with data. Science of Computer Programming 56(3), 251–273 (2005)
17. Groote, J.F., Willemse, T.A.C.: Parameterised boolean equation systems. Theoretical Computer Science 343(3), 332–369 (2005)
18. Kwak, H., Choi, J., Lee, I., Philippou, A.: Symbolic weak bisimulation for value-passing calculi. Technical Report, MS-CIS-98-22, Department of Computer and Information Science, University of Pennsylvania (1998)
19. Lin, H.: Symbolic transition graph with assignment. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 50–65. Springer, Heidelberg (1996)
20. Lynch, N., Tuttle, M.: An introduction to input/output automata. CWI Quarterly 2(3), 219–246 (1989)
21. Mader, A.: Verification of modal properties using boolean equation systems. PhD Thesis, VERSAL 8, Bertz Verlag, Berlin (1997)
22. Mateescu, R.: A generic on-the-fly solver for alternation-free boolean equation systems. In: Garavel, H., Hatcliff, J. (eds.) ETAPS 2003 and TACAS 2003. LNCS, vol. 2619, pp. 81–96. Springer, Heidelberg (2003)
23. Milner, R.: A Calculus of Communicating Systems. Springer, Heidelberg (1980)
24. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)

25. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes (Part I/II). Information and Computation 100(1), 1–77 (1992)
26. Paige, R., Tarjan, R.: Three partition refinement algorithms. SIAM Journal of Computing 16(6), 973–989 (1987)
27. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5(2), 285–309 (1955)
28. Zhang, D., Cleaveland, R.: Fast generic model-checking for data-based systems. In: Wang, F. (ed.) FORTE 2005. LNCS, vol. 3731, pp. 83–97. Springer, Heidelberg (2005)

Decidability Results for Well-Structured Transition Systems with Auxiliary Storage

R. Chadha[⋆] and M. Viswanathan[⋆⋆]

Dept. of Computer Science, University of Illinois at Urbana-Champaign

Abstract. We consider the problem of verifying the safety of well-structured transition systems (WSTS) with auxiliary storage. WSTSs with storage are automata that have (possibly) infinitely many control states along with an auxiliary store, but which have a well-quasi-ordering on the set of control states. The set of reachable configurations of the automaton may themselves not be well-quasi-ordered because of the presence of the extra store. We consider the coverability problem for such systems, which asks if it is possible to reach a control state (with some store value) that covers some given control state. Our main result shows that if control state reachability is decidable for automata with some store and *finitely* many control states then the coverability problem can be decided for WSTSs (with infinitely many control states) and the same store, provided the ordering on the control states has some special property. The special property we require is defined in terms of the existence of a ranking function compatible with the transition relation. We then show that there are several classes of infinite state systems that can be viewed as WSTSs with an auxiliary storage. These observations can then be used to both reestablish old decidability results, as well as discover new ones.

1 Introduction

Algorithmic verification of infinite state systems has received considerable attention from the research community in the past decade because the semantics of many systems can be naturally described using an unbounded state space. Examples of such systems include recursive software (sequential or concurrent, with or without dynamic allocation), asynchronous distributed systems, real-time systems, hybrid systems, and stochastic systems. Since the general problem of model checking such systems is known to be undecidable, a variety of solutions have been proposed. These include semi-decision procedures to verify a system [5,3,25] or to find bugs [23,6], as well as identifying special classes of infinite state systems (and properties) for which model checking can be shown to be decidable [22,21,4,11,7,10,24].

⋆ Supported by DARPA/AFOSR MURI award F49620-02-1-0325 and NSF CCF 0429639.

⋆⋆ Supported by NSF CCF 0448178 and NSF CCF 0509321.

While specialized approaches have been used to prove positive decidability results in many cases, a few general and broad techniques have emerged. One important technique is the use of *well-quasi-orders (wqo)* [17] (or stronger notions like *better-quasi-orders* [4]). The idea here is to identify a simulation relation (or some variant, like weak simulation, or stuttering simulation) on the (infinite state) transition system which is also a wqo. Then using the observation that any increasing sequence (with respect to subset ordering) of upward closed sets (with respect to a wqo) eventually stabilizes, a variety of problems, like backward reachability and simulation by finite state processes, can be shown to be decidable [1,11]. Transition systems with a wqo simulation relation are called *well-structured transition systems (WSTS).* Examples of WSTSs can be found in [1,11].

In this paper, we ask when the approach of using w.q.o.s can be combined with other techniques to prove general decidability results. We consider the problem of verifying safety properties of well-structured transition systems (WSTS) with an auxiliary store. WSTSs with auxiliary storage, which we call w.q.o. *automata*, are automata with (possibly infinitely many) control states that can store and retrieve information from an auxiliary data structure. Formally a data store is a domain of possible values, along with a set of predicates and operations to transform the store. The transitions of a w.q.o. automaton are guarded by a (predefined) predicate to test the value of the store, and transform the store using an allowed operation, in addition to changing the control state. Such automata are called w.q.o. automata because the control states are required to be ordered by a well-quasi-ordering that is compatible with the transition relation — a state q can be simulated by all control states greater (with respect to the ordering on states) than q. The semantics of such an automaton can be defined using a transition system, where the configurations are pairs (q, d), of a control state q, and a data value d. Notice the natural ordering on configurations — $(q_1, d_1) \preceq (q_2, d_2)$ iff $q_1 \preceq q_2$ and $d_1 = d_2$ — is not in general a w.q.o., and moreover, there maybe no w.q.o. on configurations compatible with the transitions. Therefore, techniques from the theory of WSTSs cannot be used directly to solve the model checking problem of wqo automata.

Our main theorem proves the decidability of the coverability problem for certain special w.q.o. automata. Recall that in the coverability problem we are given a control state q, and asked whether there is an execution, starting from the initial configuration, that can reach some control state $q' \succeq q$. The conditions required to prove decidability are as follows. First we require that the control state reachability problem be decidable for automata with *finitely many control states* and the same data store. Second, we require a ranking function on states compatible with the w.q.o. on states such that the number of states with a bounded rank (for any bound k) is finite. Finally, we require that transitions of the wqo automata that decrease the rank of the control state, are enabled only at a fixed data store and do not change the data store. We show that if a *wqo* automata satisfies these conditions then a backward reachability algorithm terminates, and can be used to solve the coverability problem; the termination of the algorithm relies on the properties of well-quasi orders. It is important to note that we have no requirements on the algorithm solving the control state reachability problem for the finite control state case (first condition above), and so it could rely on any of the techniques that have been discovered in the past.

We then show that there are many natural classes of systems that can be viewed as w.q.o. automata, for which our decidability result applies. Our main result can then be used to rediscover old results (but with a new proof), and establish many new decidability results. We present herein two applications of the result, while some others may be found in [8]. First we consider asynchronous programs [15,12,14,18,19], which are recursive programs that make both conventional *synchronous* function calls, where a caller waits until the callee completes computation, and *asynchronous* procedure calls, which are not immediately executed but are rather stored and "dispatched" by an external scheduler at a later point. Such systems can be abstracted (using standard techniques like predicate abstraction [13]) into automata with a multi-set (to store pending asynchronous calls) and a stack (for recursive calls), but which remove elements from the multi-set only when the stack is empty. The control state reachability problem for such recursive multi-set automata has been previously shown to be decidable [24,16], and our main theorem provides a new proof of this fact. Moreover, because our main theorem is a generalization of these results, it can also be used to establish new decidability results. In particular we can prove the decidability of the control state reachability problem for automata with both a multi-set and a higher-order stack [7]. Such automata can be used to model asynchronous programs, where asynchronous procedures can be more generally (safe) higher-order recursive programs, rather than (first-order) recursive procedures.

Asynchronous programming as an idiom is being widely used in a variety of contexts. One particular context is that of networked embedded systems [12,14], where asynchronous procedure calls form the basis of event-driven programming languages. Such embedded systems often need to meet real-time constraints, and so the dispatcher is required to schedule pending asynchronous calls based on the time when they were invoked. We show that such programs with boolean variables [1], can be modeled by automata with a stack, and multi-set of clocks. Then using our main theorem we prove the decidability of the control state reachability problem for such systems.

Paper Outline. The rest of the paper is organized as follows. First we discuss closely related work. Then in Section 2, we present basic definitions and properties of well-quasi orders and ranking functions. We formally define w.q.o. automata in Section 3. Our main decidability result is presented next (Section 4). Section 5, gives examples of w.q.o. automata, and discusses the consequences of our main decidability result. Finally we conclude (Section 6) with some observations and future work. For lack of space reasons, we shall omit the proofs which may be found in an accompanying technical report [8].

[1] A general program can always be abstracted using techniques such as predicate abstraction [13] to obtain such restricted programs.

1.1 Related Work

There is a large body of work on infinite state verification, and we cannot hope to justice to them in such a paper; therefore this section limits itself to work that is very close in spirit to this paper. Our work continues a line of work started in [24], where we considered automata with multi-sets and stacks to model asynchronous programs. The decidability of the control state reachability problem was proved using w.q.o. theory and Parikh's theorem. In [16], Jhala and Majumdar, simplified the proof, removing its reliance on Parikh's theorem. However both these proofs use features that are very specific to multi-sets and stacks, and cannot be easily generalized to obtain verification algorithms for the models considered in this paper. In particular, our original motivation was to look at the problem of verifying networked embedded systems [12,14], which are real-time asynchronous programs, and the proof techniques in [24,16] do not generalize to such a model. We discuss the differences between our proof approach and then one in [16] in more detail, when we present the main theorem.

Another very closely related work is the paper by Emmi and Majumdar [9]. One of the main observations concerns w.q.o. pushdown automata, which are pushdown automata with infinitely many control states that have a well-quasi ordering on control states. They show the decidability of the control state sub-covering problem for such automata. Unfortunately, the proof presented in the paper is incorrect [20]. The authors conjecture that the decidability result for w.q.o. pushdown automata is true. Even if the conjecture is successfully proved there are some differences with our main theorem. First, the Emmi-Majumdar result considers *downward compatibility* of the ordering with the transitions and the sub-covering problem, whereas we consider upward compatibility and the coverability problem. Next, their result specifically applies to automata with stacks, and not to other data structures like higher-order stacks that we consider here. On the flip side, their conjecture does not impose any conditions on the wqo on states itself (like ranking functions) that we require for our result. So if the Emmi-Majumdar conjecture is successfully proved then the main theorem here apply to incomparable classes of systems.

2 Preliminaries

Well-quasi-orders. A binary relation $\preceq$ on a set Q is said to be a *pre-order* if $\preceq$ is reflexive and transitive. Please note that a pre-order need not satisfy anti-symmetry, *i.e.*, it may be the case that $q \preceq q'$ and $q' \preceq q$ for $q \neq q'$. We shall say that q *is strictly less that* q' *(written as* $q \prec q'$*)* if $q \preceq q'$ but $q' \not\preceq q$. Two elements q, q' are said to be *comparable* if either $q \preceq q'$ or $q' \preceq q$ and said to be *incomparable* otherwise. We write $q \succeq q'$ if $q' \preceq q$ and $q \succ q'$ if $q' \prec q$.

A pre-order $\preceq$ on a set Q is said to be a *well-quasi-order* if every countably infinite sequence of elements $q_1, q_2, q_3, \ldots,$ from Q contains elements $q_r \preceq q_s$ for some $0 \leq r < s$. Equivalently, a pre-order is a well-quasi-order if there is no infinite sequence of pairwise incomparable elements and there is no strictly descending infinite sequence (of the form $q_1 \succ q_2 \succ q_3 \succ \ldots$). For the rest of paper, we shall say that $(\mathsf{Q}, \preceq)$ is a $\mathsf{w.q.o.}$ if $\preceq$ is a well-quasi-order on Q.

Given a $\mathsf{w.q.o.}$ $(\mathsf{Q}, \preceq)$ and $\mathsf{Q}' \subseteq \mathsf{Q}$, we say that $\mathsf{M}_{Q'} \subseteq \mathsf{Q}'$ is a *minor set* for Q' if i) for all $q \in \mathsf{Q}'$ there is a $q' \in \mathsf{M}_{Q'}$ such that $q' \preceq q$, and ii) for all $q_1, q_2 \in \mathsf{M}_{Q'}$, $q_1 \neq q_2$ implies $q_1 \not\preceq q_2$. The definition of well-quasi-ordering implies that each subset of Q has at least one minor set and all minor sets are finite.

A set $\mathsf{U} \subseteq \mathsf{Q}$ is said to be *upward closed* if for every $q_1 \in \mathsf{U}$ and $q_2 \in \mathsf{Q}$, $q_1 \preceq q_2$ implies that $q_2 \in \mathsf{U}$. An upward closed set is completely determined by its minor set: if M_U is a minor set for U then $\mathsf{U} = \{q \in \mathsf{U} \,|\, \exists q_m \in \mathsf{M}_U \text{ s.t. } q_m \preceq q\}$. Also any subset $\mathsf{Q}' \subseteq \mathsf{Q}$ determines an upward closed set, $\mathsf{U}_{\mathsf{Q}'} = \{q \,|\, \exists q' \in Q' \text{ s.t. } q' \preceq q\}$. The following important observation follows from $\mathsf{w.q.o.}$ theory.

Proposition 1. *For every infinite sequence of upward closed sets* $\mathsf{U}_1, \mathsf{U}_2 \ldots\ldots$ *such that* $\mathsf{U}_r \subseteq \mathsf{U}_{r+1}$ *there is a* j *such that* $\mathsf{U}_l = \mathsf{U}_j$ *for all* $l \geq j$.

Ranking functions. If the order $\preceq$ also satisfies anti-symmetry then it is possible to define a function rank from Q into the class of ordinals as: $\mathsf{rank}(q) = 0$ if Q does not have any elements strictly less than q and $\mathsf{rank}(q) = sup(\{\mathsf{rank}(q') \,|\, q' \prec q\}) + 1$ otherwise. The function rank guarantees that if q_1, q_2 are comparable then $\mathsf{rank}(q_1) < \mathsf{rank}(q_2)$ iff $q_1 \prec q_2$. We adapt the concept of the rank function for pre-orders. First instead of working with the whole class of ordinals, we shall work with the set of natural numbers[2]. Furthermore, we shall only require that $\mathsf{rank}(q_1) \leq \mathsf{rank}(q_2)$ if $q_1 \preceq q_2$.

Definition 1. *[Ranking function] Given a* $\mathsf{w.q.o.}$ $(\mathsf{Q}, \preceq)$, *a function* $\alpha : \mathsf{Q} \to \mathbb{N}$ *is said to be a ranking function if for every* $q_1, q_2 \in \mathsf{Q}$, $q_1 \preceq q_2$ *implies* $\alpha(q_1) \leq \alpha(q_2)$.

The ranking function α can be extended to a function on the set of upward closed sets of $(\mathsf{Q}, \preceq)$ as follows. Let M_U be a minor set for an upward closed U. Let $\alpha_{\max}(\mathsf{U}) = \max\{\alpha(q) | q \in \mathsf{M}_\mathsf{U}\}$. It can be easily shown that this extension is well-defined, *i.e.*, does not depend on the choice of M_U.

3 Well-Structured Transition Systems with Auxiliary Storage

The paper considers automata with an auxiliary store and possibly infinitely many control states, such that there is a well-quasi-ordering defined on the control states. We formally, define such automata in this section. We begin by first introducing the concept of a pointed data structure that formalizes the notion of an auxiliary store.

3.1 Pointed Data Structures

Definition 2 (Pointed data structure). *A pointed data structure is a tuple* $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, d_i, p_i)$ *such that* D *is a set,* $\widetilde{op}$ *is a collection of functions*

[2] The set of natural numbers is also the set of all finite ordinals.

$f : D \to D$, $\widetilde{pred}$ is a collection of unary predicates on D, d_i is an element of D and $p_i \in \widetilde{pred}$ is a unary predicate on D such that $p_i(d) \Leftrightarrow (d = d_i)$. The elements of D are henceforth called data values and the data value d_i is said to be the initial data value.

For example, a pushdown store on a alphabet Γ in a pushdown automata can be formalized as follows. The set Γ^* (set of all finite strings over Γ) can be taken as the set of data values with the empty string ϵ as the initial value. The set of predicates $\widetilde{pred}$ can be chosen as $\{\mathsf{empty}\} \cup \{\mathsf{top}_\gamma \mid \gamma \in \Gamma\} \cup \{\mathsf{any}\}$, where $p_i = \{\epsilon\}$ (the initial predicate), $\mathsf{top}_\gamma = \{w\gamma \mid w \in \Gamma^*\}$ (the top of stack is γ) and $\mathsf{any} = \Gamma^*$ (any stack). The set of functions $\widetilde{op}$ can be defined as $\{id\} \cup \{\mathsf{push}_\gamma \mid \gamma \in \Gamma\} \cup \{\mathsf{pop}_\gamma \mid \gamma \in \Gamma\}$ where push_γ and pop_γ are defined as follows. For all $w \in \Gamma^*$, $\mathsf{push}_\gamma(w) = w\gamma$ and $\mathsf{pop}_\gamma(w) = w_1$ if $w = w_1\gamma$ and w otherwise. In a pushdown system the functions pop_γ will be enabled only when the store satisfies top_γ and any respectively.

For the rest of paper, we shall assume that our data structure has a finite number of predicates and a finite number of functions. This is mainly a matter of convenience and we could have dealt with countable number of predicates and functions as long as the data structure is finitely presentable. We could also have dealt with a finite number of initial values, partial functions and relations in the set $\widetilde{op}$. However, for the sake of clarity, we have omitted these cases here.

3.2 w.q.o. Automata

We now formalize the notion of WSTS with auxiliary storage.

Definition 3. *[w.q.o. automaton] A w.q.o. automaton on a pointed data structure $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, d_i, p_i)$ is a tuple $(\mathsf{Q}, \preceq, \delta, q_i)$ such that*

1. *$(\mathsf{Q}, \preceq)$ is a w.q.o.,*
2. *$q_i \in \mathsf{Q}$ and*
3. *$\delta \subseteq \mathsf{Q} \times \widetilde{pred} \times \widetilde{op} \times \mathsf{Q}$. Furthermore, the set δ is upward compatible, i.e. for every $(q, p, g, q') \in \delta$ and $q \preceq q_1$ there exists q'_1 such that $q' \preceq q'_1$ and $(q_1, p, g, q'_1) \in \delta$.*

The set Q is said to be the set of control states of the automaton, δ the transition function and q_i the initial state. If the set Q is finite then we say that it is a finite w.q.o. automaton.

Given an upward closed set $\mathsf{U} \subseteq \mathsf{Q}$, a predicate $p \in \widetilde{pred}$, $g \in \widetilde{op}$ let $\delta^{-1}(p, g, \mathsf{U})$ be the set $\{q \mid \exists q' \in \mathsf{U} \text{ s.t. } (q, p, g, q') \in \delta\}$. The upward compatibility of δ ensures that $\delta^{-1}(p, g, \mathsf{U})$ is either empty or upward closed.

An example of a *wqo* automaton in literature is multi-set pushdown automata [24,16] and discussed in Section 5.1. For the remainder of the section, we shall fix a pointed data structure $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, d_i, p_i)$ and a w.q.o. automaton $\mathbf{A}$ on D.

The semantics of a *wqo* automaton is defined in terms of a transition system over a set of configurations. A *configuration* is a pair (q, d), where $q \in \mathsf{Q}$ is a control state and $d \in D$ is a data value. The pair (q_i, d_i) is said to be the *initial configuration.* For $\delta_0 \subseteq \delta$, we say $(q_1, d_1) \rightarrow_{\mathbf{A},\delta_0} (q_2, d_2)$ if there is a transition $(q_1, p, g, q_2) \in \delta_0$ such that $p(d_1)$ and $d_2 = g(d_1)$. We shall omit $\mathbf{A}$ if the automaton under consideration is clear from the context. The transition relation on configurations will be $\rightarrow_{\mathbf{A},\delta}$. The n-fold composition of $\rightarrow_{\delta_0}$ will be denoted by $\rightarrow^n_{\delta_0}$. In other words, $(q, d) \rightarrow^n_{\delta_0} (q', d')$ iff there are $(q_0, d_0), (q_1, d_1), \ldots (q_n, d_n)$ such that $q_0 = q, d_0 = d$, $q_n = q', d_n = d'$ and for every $0 \leq r < n$ $(q_r, d_r) \rightarrow_{\delta_0} (q_{r+1}, d_{r+1})$. We shall write $(q, d) \rightarrow^*_{\delta_0} (q', d')$ if there is some $j \geq 0$ such that $(q, d) \rightarrow^j_{\delta_0} (q', d')$. Finally, we shall say that the configuration (q', d') is reachable from (q, d) using transition in δ_0 if $(q, d) \rightarrow^*_{\delta} (q', d')$.

Given a set $\mathsf{Q}' \subseteq Q$, it will be useful to define two sets, $\mathsf{Pre}^*_{\mathbf{A},\,\delta_0,d_i}(Q')$ and $\mathsf{Pre}^*_{\mathbf{A},\,\delta_0}(Q')$. The set $\mathsf{Pre}^*_{\mathbf{A},\,\delta_0,d_i}(\mathsf{Q}') = \{q \,|\, \exists q' \in \mathsf{Q}' \text{ s.t. } (q, d_i) \rightarrow^*_{\delta_0} (q', d_i)\}$ gives the set of all states from which some control state in Q' can be reached starting with and ending with the initial data value d_i. The set $\mathsf{Pre}^*_{\mathbf{A},\,\delta_0}(\mathsf{Q}') = \{q \,|\, \exists q' \in \mathsf{Q}', d \in D \text{ s.t. } (q, d_i) \rightarrow^*_{\delta_0} (q', d)\}$ gives the set of all states from which some control state in Q' can be reached starting with the initial data value and ending with some data value. We will omit the subscript $\mathbf{A}$ if it is clear from the context.

Observe that since δ is upward compatible, if $(q, d) \rightarrow^n_{\delta} (q', d')$ then for every $q_1 \succeq q$ there exists an $q'_1 \succeq q'$ such that $(q_1, d) \rightarrow^n_{\delta} (q'_1, d')$. Therefore, for any upward-closed set U, the sets $\mathsf{Pre}^*_{\mathbf{A},\,\delta}(\mathsf{U})$ and $\mathsf{Pre}^*_{\mathbf{A},\,\delta,d_i}(\mathsf{U})$ are upward closed.

It will also be useful to identify two kinds of transitions in a *wqo* automaton. The first ones are fired only when the data is initial and preserve the data.

Definition 4 (Initial data preserving). *A transition $(q, p, g, q') \in \delta)$ is said to be initial data preserving if $p = p_i$ and $g = id$ where id is the identity function.*

The other kind of transitions will be *rank non-decreasing* transitions which will be defined with respect to a ranking function (see Definition 1).

Definition 5 (Rank non-decreasing). *Given a ranking function α on $(\mathsf{Q}, \preceq)$ and a set of transition $\delta_0 \subseteq \delta$, we say that δ_0 is rank α non-decreasing if for each predicate $p \in \widetilde{pred}$, function $g \in \widetilde{op}$ and upward closed set $\mathsf{U} \subset \mathsf{Q}$ either $\delta_0^{-1}(p, g, \mathsf{U}) = \emptyset$ or $\delta_0^{-1}(p, g, \mathsf{U})$ is upward-closed and $\alpha_{\max}(\delta_0^{-1}(p, g, \mathsf{U})) \leq \alpha_{\max}(\mathsf{U})$.*[3]

As mentioned in the introduction, we will consider special w.q.o. automata, namely those in which the only rank decreasing transitions are those that are also initial data preserving. We will say that such a restricted w.q.o. automata is *compatible to a ranking function*; we define this formally next.

Definition 6 (Compatibility of ranking function). *Given a ranking function α on $(\mathsf{Q}, \preceq)$ we say that α is compatible with δ if*

1. $\alpha(q_i) = 0$, and

[3] Please note that our definition of rank-decreasing functions is different from standard one.

2. there exist $\delta = \delta_a \cup \delta_b$ such that
   - *δ_b is the set of all initial data preserving transitions.*
   - *$\delta_a = \delta \setminus \delta_b$ and δ_a is rank α non-decreasing. The pair (δ_a, δ_b) is said to be the α-compatible splitting of δ.*

The condition $\alpha(q_i) = 0$ is not a serious constraint as we can always define a ranking function $\alpha'(q) = \max(\alpha(q) - \alpha(q_i), 0)$. The second condition implies that the transition function δ can be split into two disjoint upward-compatible sets δ_a and δ_b. All transitions in δ_b are enabled only when the data value is initial and do not modify the data. This condition on δ_b ensures that any computation $(q, d_i) \rightarrow^*_\delta (q', d')$ is of the form $(q, d_i) \rightarrow^*_{\delta_a} (q_0, d_i) \rightarrow_{\delta_b} (q_1, d_i) \rightarrow^*_{\delta_a} (q_2, d_i) \rightarrow_{\delta_b} (q_2, d_i) \ldots \rightarrow_{\delta_b} (q_n, d_i) \rightarrow^*_{\delta_a} (q', d')$ for some $q_0, q_1, \ldots q_n \in \mathsf{Q}$.

4 Decidability of the Coverability Problem

Given a $\widetilde{\mathsf{w.q.o.}}$ automaton $\mathbf{A} = (\mathsf{Q}, \preceq, \delta, q_i)$ on a pointed data structure $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, d_i, p_i)$, an upward closed set U, we are interested in deciding if there is some configuration (q, d) such that $q \in \mathsf{U}$ and q is reachable from the initial configuration q_i. The upward closed set U is often represented by its finite minor set. This problem is known as *coverability problem* and we shall study two versions of this problem. The first is whether there exists some state $q \in \mathsf{U}$ such that the configuration (q, d_i) is reachable from (q_i, d_i), *i.e.*, if $q_i \in \mathsf{Pre}^*_{\mathsf{A},\, \delta, d_i}(\mathsf{U})$. Secondly whether there exists some state $q \in \mathsf{U}$ and $d \in D$ such that (q, d) is reachable from (q_i, d_i), *i.e.*, if $q_i \in \mathsf{Pre}^*_{\mathsf{A},\, \delta}(\mathsf{U})$.

We start by describing how the coverability problem is tackled in WSTSs without the store. In that case, the transition relation $\delta \subseteq \mathsf{Q} \times \mathsf{Q}$ and the coverability problem is equivalent to deciding whether the reflexive transitive closure $(\delta^{-1})^*$ of the relation δ^{-1} contains q_i or not. A backward reachability analysis is performed, and an increasing sequence U_i is generated such that $\mathsf{U}_1 = \mathsf{U}$ and $\mathsf{U}_{j+1} = \mathsf{U}_j \cup \delta^{-1}(\mathsf{U}_j)$. The upward compatibility ensures that $\delta^{-1}(\mathsf{U}_j)$ is upward closed. Since any increasing sequence of upward closed sets in a w.q.o. must be eventually constant, we terminate once $\mathsf{U}_j = \mathsf{U}_{j+1}$.

Our approach to the coverability problem for a w.q.o. automaton will follow a similar approach. Given a $\widetilde{\mathsf{w.q.o.}}$ automaton $\mathbf{A} = (\mathsf{Q}, \preceq, \delta, q_i)$ on a pointed data structure $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, d_i, p_i)$, we shall assume the existence of a ranking function α compatible with the transition relation δ. Let (δ_a, δ_b) be the α-splitting of δ. We shall always assume that we can compute the minor set for $\delta_b^{-1}(\mathsf{U}, p_i, id)$ given a minor set for U. However, we shall need to compute $\mathsf{Pre}^*_{\mathbf{A},\, \delta_a, d_i}(\mathsf{U})$. For this we shall need the notion of rank k-approximations.

4.1 Rank k-Approximations

Intuitively, the rank k-approximation $\mathbf{A}_k$ of $\mathbf{A}$ is constructed from the subset of control states of $\mathbf{A}$ whose rank is less than k. The construction is carried out carefully in order to capture all the computations of the original automaton that use the transitions in δ_a.

Definition 7 (Rank k-approximation). *Given a w.q.o. automaton $\mathbf{A} = (\mathsf{Q}, \preceq, \delta, q_i)$ on a pointed data structure $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, d_i, p_i)$, a ranking function α on $(\mathsf{Q}, \preceq)$ such that α is compatible with δ. Let (δ_a, δ_b) be the α-compatible splitting of δ. Given $k \in \mathbb{N}$, the rank k approximation is defined as the automaton $\mathbf{A}_k = (\mathsf{Q}_{\leq k}, \preceq_k, \delta_k, q_i)$ where:*

- $\mathsf{Q}_{\leq k} = \{q \in \mathsf{Q} \mid \alpha(q) \leq k\}$,
- *$q \preceq_k q'$ for $q, q' \in \mathsf{Q}_{\leq k}$ iff $q \preceq q'$ and*
- *$(q, p, g, q') \in \delta_k$ for $q, q' \in \mathsf{Q}_{\leq k}$ iff there exists $q'' \in \mathsf{Q}$ such that $q'' \succeq q'$ and $(q, p, g, q'') \in \delta_a$.*

For the rest of the section, we shall assume a fixed w.q.o. automaton $\mathbf{A}$=$(\mathsf{Q}, \preceq, \delta, q_i)$ on a fixed pointed data structure $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, d_i, p_i)$ and a fixed ranking function α on $(\mathsf{Q}, \preceq)$ compatible with δ. Let (δ_a, δ_b) be the α-compatible splitting of the transition relation δ.

The following Lemma states that any computation in the rank k-approximation of $\mathbf{A}$ corresponds to a computation of $\mathbf{A}$ that uses the transitions in δ_a.

Lemma 1 (Soundness of approximation). *Let $\mathbf{A}_k = (\mathsf{Q}, \preceq, \delta, q_i)$ be the rank k-approximation of $\mathbf{A}$. For any $q, q_1 \in \mathsf{Q}_{\leq k}, d, d_1 \in D$, if $(q, d) \rightarrow^*_{\delta_k} (q_1, d_1)$ then there is some $q'_1 \in \mathsf{Q}$ such that $q'_1 \succeq q_1$ and $(q, d) \rightarrow^*_{\delta_a} (q'_1, d_1)$.*

The following Lemma states that any computation of the automaton $\mathbf{A}$ that uses transitions in δ_a and ending in a state which is above some state $q_0 \in \mathsf{Q}_{\leq k}$ is reflected in its k-th approximation.

Lemma 2 (Faithfulness of approximation). *Let $\mathbf{A}_k = (\mathsf{Q}, \preceq, \delta, q_i)$ be a rank k-approximation of $\mathbf{A}$. If there are $q_0, q, q' \in \mathsf{Q}$ such that $q_0 \in \mathsf{Q}_{\leq k}$, $q' \succeq q_0$ and $(q, d) \rightarrow^*_{\delta_a} (q', d')$, then there is a $q_1 \in \mathsf{Q}_{\leq k}$ such that $q_1 \preceq q$ and $(q_1, d) \rightarrow^*_{\delta_k} (q_0, d')$.*

The above two lemmas show that if $\mathsf{U} \subseteq \mathsf{Q}$ is an upward closed set such that $\alpha_{\max}(\mathsf{U}) = k$ then minor sets for the sets $\mathsf{Pre}^*_{\mathbf{A}, \delta_a, d_i}(\mathsf{U})$ and $\mathsf{Pre}^*_{\mathbf{A}_k, \delta_k}(U)$ are the minor sets for $\mathsf{Pre}^*_{\mathbf{A}_k, \delta_k, d_i}(\mathsf{Q}')$ and $\mathsf{Pre}^*_{\mathbf{A}_k, \delta_k}(Q')$ respectively. Thus, if we have algorithms to compute minor sets for $\mathsf{Pre}^*_{\mathbf{A}_k, \delta_k, d_i}(\mathsf{Q}')$ and $\mathsf{Pre}^*_{\mathbf{A}_k, \delta_k}(Q')$ for any subset $Q' \subseteq Q$, then we can compute the minor sets $\mathsf{Pre}^*_{\mathbf{A}, \delta_a, d_i}(\mathsf{U})$ and $\mathsf{Pre}^*_{\mathbf{A}, \delta_a}(\mathsf{U})$ for an upward closed set U whose rank is k. Towards this end, we shall define effective *wqo* automata.

4.2 Effective w.q.o.Automata and Coverability

We start by defining a ranking function effectively compatible with the transition relation of a *wqo* automata.

Definition 8 (Effectively compatible ranking functions). *Given a w.q.o. automaton, $\mathbf{A} = (\mathsf{Q}, \leq, \delta, q_i)$ on the pointed data structure $\mathsf{D} = (D, \widetilde{pred}, \widetilde{op}, p_i, d_i)$, a ranking function $\alpha : \mathsf{Q} \rightarrow \mathbb{N}$ compatible with δ is said to be effectively compatible with δ if the following hold.*

– *For each $k \in \mathbb{N}$, the set $Q_{\leq k} = \{q \in \mathsf{Q} \mid \alpha(q) \leq k\}$ is finite.*
– *There is an algorithm* Rank *that on input $q \in \mathsf{Q}$ computes $\alpha(q)$.*
– *There is an algorithm* Elements *that on input $k \in \mathbb{N}$ computes the set $\mathsf{Q}_{\leq k}$.*
– *There is an algorithm* Approx *that on input $k \in \mathbb{N}$ outputs the rank k-approximation.*[4]

An effectively compatible ranking function is finitely presented by the algorithms Rank*,* Elements *and* Approx*.*

If α is effectively compatible with δ, and the relation $\preceq$ is decidable then Lemma 1 and Lemma 2 ensure that algorithms to check if $q_1 \in \mathsf{Pre}^*_{\mathbf{A},\delta_a,d_i}(\mathsf{U})$ or $q_1 \in \mathsf{Pre}^*_{\mathbf{A},\delta_a}(U)$ exist if state reachability can be solved for finitely many states. We are now almost ready to prove our main theorem. We need one more definition.

Definition 9 (Effective w.q.o. automaton). *A w.q.o. automaton $\mathbf{A} = (\mathsf{Q}, \leq, \delta, q_i)$ on the pointed data structure $\mathsf{D} = (D, \widetilde{pred}, \widetilde{op}, p_i, d_i)$ is said to be effective if the following hold.*

– *There is a ranking function $\alpha : \mathsf{Q} \rightarrow \mathbb{N}$ effectively compatible with δ. Let the algorithms* Rank*,* Elements*,* Approx *finitely present the ranking function.*
– *There is an algorithm* Less *that on inputs q_1 and q_2 returns true is $q_1 \preceq q_2$ and false otherwise.*
– *There is an algorithm* InvDeltab *which given a minor set for an upward closed set U returns a minor set for $\delta_b^{-1}(\mathsf{U}, p_i, id)$ where id is the identity function on D where δ_b is the set of initial data preserving transitions.*

An effective automaton is finitely presented by the algorithms Rank*,* Elements*,* Approx*,* Less *and* InvDeltab*.*

We now have the main result of the paper.

Theorem 1 (Decidability of the coverability problem). *Assume that for a data structure $\mathsf{D} = (D, \widetilde{pred}, \widetilde{op}, p_i, d_i)$ there are two algorithms $\mathcal{A}$ and $\mathcal{B}$ such that the following hold.*

– *Given a finite wqo automaton $\mathbf{A}_1 = (\mathsf{Q}_1, \preceq_1, \delta_1, q_i)$, and $q_1, q_1' \in Q_1$ the algorithm $\mathcal{A}$ returns true if $q_1 \in \mathsf{Pre}^*_{\mathbf{A}_1,\delta_1,d_i}(q_1')$ and false otherwise.*
– *Given a finite wqo automaton $\mathbf{A}_1 = (\mathsf{Q}_1, \preceq_1, \delta_1, q_i)$, and $q_1, q' \in Q_1$, the algorithm $\mathcal{B}$ returns true if $q_1 \in \mathsf{Pre}^*_{\mathbf{A}_1,\delta_1}(q_1')$ and false otherwise.*

Let $\mathbf{A} = (Q, \widetilde{pred}, \widetilde{op}, q_i)$ be an effective (not necessarily finite) automaton. Given a minor set M_U for an upward closed set U, there is an algorithm to decide if there is some $q \in \mathsf{U}$ such that the configuration (q, d_i) is reachable from the initial configuration (q_i, d_i). Similarly there is an algorithm to decide if there is some $q \in \mathsf{U}$ and some $d \in \mathsf{D}$ such that (q, d) is reachable from the initial configuration (q_i, d_i).

[4] Please note that since we are assuming that predicates and functions are finite sets, a finite presentation of the rank k-function always exists.

Proof. Let α be the ranking function effectively compatible with δ. Let (δ_a, δ_b) be the α-compatible splitting of δ. In order to decide whether $q_i \in \mathsf{Pre}^*_{\mathsf{A},\,\delta,d_i}(\mathsf{U})$, we construct the increasing sequence $\mathsf{U}_0 \subseteq \mathsf{U}_1 \subseteq \mathsf{U}_2, \ldots$ such that $\mathsf{U}_0 = \mathsf{U}$ and $\mathsf{U}_{r+1} = \mathsf{U}_r \cup \mathsf{Pre}^*_{\mathbf{A},\,\delta_a,d_i}(\mathsf{U}_r \cup \delta_b^{-1}(\mathsf{U}_r, p_i, id))$ where id is the identity function.

As (δ_a, δ_b) is an α-compatible splitting of δ, any computation $(q, d_i) \to^*_\delta (q', d')$ is of the form $(q, d_i) \to^*_{\delta_a} (q_0, d_i) \to_{\delta_b} (q_1, d_i) \to^*_{\delta_a} (q_2, d_i) \to_{\delta_b} \ldots \to_{\delta_b} (q_n, d_i) \to^*_{\delta_a} (q', d')$ for some $q_0, q_1, \ldots q_n \in \mathsf{Q}$. Thus, $\mathsf{Pre}^*_{\mathbf{A},\,\delta,d_i}(\mathsf{U}) = \bigcup_{r\geq 0} \mathsf{U}_r$. Again, as every increasing sequence of increasing upward closed sets must be eventually constant, we can terminate once we get $\mathsf{U}_r = \mathsf{U}_{r+1}$. Hence, the first question can now be answered by deciding whether $q_i \in \mathsf{U}_r$. The second question can be reduced to the first problem by considering $\mathsf{U}' = \mathsf{Pre}^*_{\mathbf{A},\,\delta_a}(\mathsf{U})$. □

Please note that the above proof is essentially different from the proof of decidability of the coverability problem in multi-set pushdown automata (MPDS) given in [16]. While we perform a backward-reachability analysis, the proof in [16] constructs a sequence of over-approximations to rule out unreachable states and a sequence of under-approximations to discover the reachable states. The proof in [16] relies essentially on the fact that the w.q.o. is formed by products of linear orders (in that case products of $\mathbb{N}$) and cannot be extended to a general w.q.o.. However, as in our case, the rank decreasing functions (decrementing of counter) are constrained and occur only when the pushdown stack is empty. We discuss the relation between the proofs in detail in [8].

5 Applications

We will now present many natural examples of w.q.o. automata. An immediate of consequence of Theorem 1, will be decidability results for safety verification for these systems. The examples that we present here, have as data store either a pushdown stack, or some variant of it. Therefore, when presenting these examples, we will use more standard notation for such stacks, rather than define them formally in terms of the domain, test predicates and operations. This mainly to make the examples easy to follow, and not clutter them with a lot of notation.

5.1 Multi-set Pushdown System

Multi-set pushdown automata have been introduced in the literature [24,16] as models of asynchronous programs that may make asynchronous procedure calls which are not immediately executed but stored and scheduled only after a recursive computation is completed.

Definition 10 (Multi-set pushdown system). *A Multi-set pushdown system (MPDS) is a tuple* $\mathbf{B} = (S, \Gamma, \Delta_{\mathsf{push}}, \Delta_{\mathsf{pop}}, \Delta_{\mathsf{cr}}, \Delta_{\mathsf{rt}}, s_0)$*, where* S *is a finite set of states,* Γ *is a finite set of stack and multi-set symbols,* $\Delta_{\mathsf{push}}, \Delta_{\mathsf{pop}}, \Delta_{\mathsf{cr}}, \Delta_{\mathsf{rt}} \subseteq S \times \Gamma \times S$ *together form the transition relation, and* s_0 *is the initial state.*

The semantics of an MPDS $\mathbf{B}$ is defined in terms of a transition system as follows. A configuration C of $\mathbf{B}$ is a tuple $(s, w, B) \in S \times \Gamma^* \times B[\Gamma]$ where Γ^*

is the set of all finite strings over Γ and $B[\Gamma]$ is the set of all multi-sets over Γ. The initial configuration of $\mathbf{B}$ is $(s_0, \epsilon, \emptyset)$ where ϵ is the empty string and $\emptyset$ is the empty multi-set. The transition relation $\Rightarrow$ on configurations is given as a union of four relations, $\Rightarrow_{\mathsf{push}}$, $\Rightarrow_{\mathsf{pop}}$, $\Rightarrow_{\mathsf{cr}}$ and $\Rightarrow_{\mathsf{rt}}$ defined as follows: $(s, w, B) \Rightarrow_{\mathsf{push}} (s', w\gamma', B)$ iff $(s, \gamma', s') \in \Delta_{\mathsf{push}}$, $(s, w\gamma, B) \Rightarrow_{\mathsf{pop}} (s', w, B)$ iff $(s, \gamma, s') \in \Delta_{\mathsf{pop}}$, $(s, w, B) \Rightarrow_{\mathsf{cr}} (s', w, B \cup \{\gamma'\})$ iff $(s, \gamma', s') \in \Delta_{\mathsf{cr}}$ and $(s, \epsilon, B \cup \{\gamma\}) \Rightarrow_{\mathsf{rt}} (s', \epsilon, B)$ iff $(s, \gamma, s') \in \Delta_{\mathsf{rt}}$. Please note that the relation Δ_{rt} assume that the stack is empty and does not change the contents of the stack. The relation $\Rightarrow^*$ is defined as the reflexive transitive closure of $\Rightarrow$.

The ordering relation $\preceq$ on $S \times B[\Gamma]$ is defined as $(s, B) \preceq (s_1, B_1)$ iff $s = s_1$ and B is a sub-multi-set of B_1; this is known to be a well-quasi-order. Using this fact, any MPDS can be seen as an instance of an *wqo* automaton with $S \times B[\Gamma]$ as the set of control states and the pushdown stack as the storage. This *wqo* automaton can be turned into an effective w.q.o. automaton by using the cardinality of the multi-set B as the ranking function. Therefore, Theorem 1 yields the following result.

Theorem 2 (Coverability of MPDS). *Given a MPDS* $\mathbf{B} = (S, \Gamma, \Delta_{\mathsf{push}}, \Delta_{\mathsf{pop}}, \Delta_{\mathsf{cr}}, \Delta_{\mathsf{rt}}, s_0)$ *and* $s \in S$*, the problem of checking whether there exist* $B \in B[\Gamma]$ *and* $w \in \Gamma^*$ *such that* $(s_0, \epsilon, \emptyset) \Rightarrow^* (s, w, B)$ *is decidable.*

The above result was proved in [24] using Parikh's theorem and in [16] using over-approximations and under-approximations; Theorem 1 yields a different and more general proof of this result. A similar result can be obtained when the asynchronous procedures are (safe) higher-order recursive procedures [8].

5.2 Timed Multi-set Pushdown System

Asynchronous programming forms the basis of event-driven languages that are used to describe networks of embedded systems [12,14]. Such systems have real-time constraints, in addition to synchronous and asynchronous procedure calls. Though the asynchronous procedure calls are postponed till the end of the execution of the current recursive procedure call, they are scheduled only if they have not passed some real-time deadline. We model this by augmenting a pushdown system with real-time clocks. When a asynchronous procedure is called, a clock is added to the multi-set and set to 0. We assume that all clocks proceed at the same rate. Once the current recursive computation is completed, the next job is scheduled only if the associated clock is within some interval bounded by integers (we assume that ∞ is an integer for this case). The results can easily be generalized to the case when the bounds are rationals rather than integers.

Definition 11. *[Timed multi-set pushdown automata] A Timed multi-set pushdown system (TMPDS) is a tuple* $\mathbf{B} = (S, \Gamma, \Delta_{\mathsf{push}}, \Delta_{\mathsf{pop}}, \Delta_{\mathsf{cr}}, \Delta_{\mathsf{rt}}, s_0)$*, where* S *is a finite set of states,* Γ *is a finite set of stack and multi-set symbols, the sets* $\Delta_{\mathsf{push}}, \Delta_{\mathsf{pop}}, \Delta_{\mathsf{cr}} \subseteq S \times \Gamma \times S$*, and* $\Delta_{\mathsf{rt}} \subseteq (S \times \Gamma \times \mathbb{N} \times \mathbb{N} \cup \{\infty\}) \times S$ *are finite and together form the transition relation, and* s_0 *is the initial state.*

The semantics of an MPDS $\mathbf{B}$ is defined in terms of a transition system as follows. A configuration C of $\mathbf{B}$ is a tuple $(s, w, B) \in S \times \Gamma^* \times B[\Gamma \times \mathbb{R}^+]$ where Γ^* is the set of all finite strings over Γ, $\mathbb{R}^+$ is the set of positive real numbers and $B[\Gamma \times \mathbb{R}^+]$ is the set of all multi-sets over $\Gamma \times \mathbb{R}^+$. The initial configuration of $\mathbf{B}$ is $(s_0, \epsilon, \emptyset)$. The transition relation $\Rightarrow$ on configurations is given as a union of five relations, $\Rightarrow_{\mathsf{push}}$, $\Rightarrow_{\mathsf{pop}}$, $\Rightarrow_{\mathsf{cr}}$,$\Rightarrow_{\mathsf{rt}}$ and $\Rightarrow_T$.

The relation $\Rightarrow_{\mathsf{push}}$ is defined as $(s, w, B) \Rightarrow_{\mathsf{push}} (s', w\gamma', B)$ iff $(s, , \gamma', s') \in \Delta_{\mathsf{push}}$. The relation $\Rightarrow_{\mathsf{pop}}$ is defined as $(s, w\gamma, B) \Rightarrow_{\mathsf{pop}} (s', w, B)$ iff $(s, \gamma, s') \in \Delta_{\mathsf{pop}}$. The relation $\Rightarrow_{\mathsf{cr}}$ is defined as $(s, w, B) \Rightarrow_{\mathsf{pop}} (s', w, B \cup \{(\gamma', 0)\})$ iff $(s, \gamma', s') \in \Delta_{\mathsf{cr}}$. The relation $\Rightarrow_{\mathsf{rt}}$ is defined as $(s, \epsilon, B \cup \{(\gamma, t)\}) \Rightarrow_{\mathsf{rt}} (s', \gamma, B)$ iff there exist $n_1, n_2 \in \mathbb{N}$ such that $n_1 \leq t \leq n_2$ and $((s, \gamma, n_1, n_2), s') \in \Delta_{\mathsf{rt}}$. The relation $\Rightarrow_T$ is defined as $(s, w, B) \Rightarrow_T (s, w, B_t)$ for every $t \in \mathbb{R}^+$ where B_t is the multi-set obtained by replacing each $(\gamma, t') \in B$ by $(\gamma, t' + t) \in B$. Observe that elements are deleted from B only when the pushdown stack is empty. The relation $\Rightarrow^*$ is defined as the reflexive transitive closure of $\Rightarrow$.

We are once again interested in an algorithm which answers the question that given $s \in S$ whether there exists $B \in B[\Gamma \times \mathbb{R}^+]$ and $w \in \Gamma^*$ such that $(s_0, \epsilon, \emptyset) \Rightarrow^* (s, w, B)$. In order to apply Theorem 1, we define a well-quasi-order $\preceq$ on $B[\Gamma \times \mathbb{R}^+]$ which gives rise to regions [2] by symmetrizing the relation $\preceq$.

The relation $\preceq$ on $B[\Gamma \times \mathbb{R}^+]$ is defined as follows. Let $n_{\max}$ be the largest natural number occurring in Δ_{cr}. Given $t \in \mathbb{R}^+$, let $\lfloor t \rfloor$ denote the integer part of t and let $frac(t)$ denote the fractional part of t. Given $B_1, B_2 \in B[\Gamma \times \mathbb{R}^+]$, we say that $B_1 \preceq B_2$ iff there exists a one-to-one function $j : B_1 \to B_2$ such that the following hold.

- If $j((\gamma_1, t_1)) = (\gamma_2, t_2)$ then $\gamma_1 = \gamma_2$.
- If $j((\gamma, t_1)) = (\gamma, t_2)$ then $t_1 \geq n_{\max}$ iff $t_2 \geq n_{\max}$.
- If $j((\gamma, t_1)) = (\gamma, t_1)$, $t_1 < n_{\max}$ then $\lfloor t_1 \rfloor = \lfloor t_2 \rfloor$ and $frac(t_1) = 0$ iff $frac(t_2) = 0$.
- If $j((\gamma, t_1)) = (\gamma, t_2)$, $j((\gamma', t_1')) = (\gamma', t_2')$ and $t_1, t_1' < n_{\max}$ then $frac(t_1) \leq frac(t_1')$ iff $frac(t_2) \leq frac(t_2')$.

This relation $\preceq$ defines a well-quasi-order on $B[\Gamma \times \mathbb{R}^+]$ [2]. The relation $\preceq$ induces an equivalence relation on $B[\Gamma \times \mathbb{R}^+]$: $B \simeq B'$ iff $B \preceq B'$ and $B' \preceq B$. An equivalence class under the relation $\simeq$ is said to be a *region*. Let $Reg(\Gamma)$ be the set of all regions defined in this way and let $Reg(B)$ be the region that contains B. The well-quasi-order $\preceq$ extends to the set of regions: $Reg(B_1) \preceq_R Reg(B_2)$ iff $B_1 \preceq B_2$. The function $\alpha_R : Reg \to \mathbb{N}$ defined as $\alpha_R(R(B)) = |B|$, where $|B|$ is the cardinality of the multi-set B, is a ranking function.

The transition relation $\Rightarrow$ can be extended to a binary relation $\Rightarrow_R$ on $S \times \Gamma^* \times Reg[\Gamma]$ as follows. We say that $(s, w, Reg(B)) \Rightarrow_R (s', w', Reg(B'))$ iff $(s, w, B) \Rightarrow (s', w', B')$. The well-defineness of the relation $\Rightarrow_R$ can be shown using the standard techniques [2]. Thus, $(s_0, \epsilon, \emptyset) \Rightarrow^* (s, w, B)$ iff $(s_0, \epsilon, Reg(\emptyset)) \Rightarrow_R^* (s, w, Reg[B])$ where $\Rightarrow_R^*$ is the reflexive transitive closure of $\Rightarrow_R$. Thus, using $S \times Reg(\Gamma)$ as the set of control states, the pushdown stack as the data structure and α_R as the ranking function, we can prove the following.

Theorem 3 (Coverability of TMPDS). *Given an TMPDS* $\mathbf{B} = (S, \Gamma, \Delta_{\mathsf{push}}, \Delta_{\mathsf{pop}}, \Delta_{\mathsf{cr}}, \Delta_{\mathsf{rt}}, s_0)$ *and* $s \in S$*, the problem of checking whether there exist* $B \in B[\Gamma \times \mathbb{R}^+]$ *and* $w \in \Gamma^*$ *such that* $(s_0, \epsilon, \emptyset) \Rightarrow^* (s, w, B)$ *is decidable.*

6 Conclusions and Future Work

We considered the coverability problem of a w.q.o. automaton which are well-structured transition systems with an auxiliary store. Our main result is that if the control state reachability problem is decidable for finite w.q.o. automaton, then there is a decision procedure based on backward-reachability analysis for the coverability problem with infinitely many states if the w.q.o. automaton satisfies certain conditions. The main requirement is the existence of a ranking function compatible with the WSTS. Intuitively, the compatibility of the ranking function ensures that rank decreasing transitions only occur when the store is the same as the initial store. For the rank non-decreasing transitions, the backward reachability is performed using the decision procedure for finite w.q.o. automaton. We showed that the decision procedure in the paper can be utilized in the contexts of asynchronous procedure calls and networked embedded systems.

We plan to implement the decision procedure in this paper and utilize it for model-checking the systems considered in this paper. A second line of research is to investigate whether other decision procedures that rely on w.q.o. theory such as the sub-covering problem can be extended to the framework of w.q.o. automaton.

References

1. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.-K.: Algorithmic analysis of programs with well quasi-ordered domains. Information and Computation 160(1), 109–127 (2000)
2. Abdulla, P.A., Jonsson, B.: Verifying networks of timed processes. In: Proceedings of the International Conference on Tools and Algorithms for Construction and Analysis of Systems, pp. 298–312 (1998)
3. Abdulla, P.A., Jonsson, B., Nilsson, M., Saksena, M.: A Survey of Regular Model Checking. In: Proceedings of the International Conference on Concurrency Theory, pp. 35–48 (2004)
4. Abdulla, P.A., Nyl'en, A.: Better is better than Well: On efficient verification of infinite state systems. In: Proceedings of the IEEE Symposium on Logic in Computer Science, pp. 132–140. IEEE Computer Society Press, Los Alamitos (2000)
5. Boigelot, B.: Symbolic Methods for Exploring Infinite State Spaces. PhD thesis, Collection des Publications de la Faculté des Sciences Appliquées de l'Université de Liége (1999)
6. Bouajjani, A., Esparza, J., Schwoon, S., Strejcek, J.: Reachability analysis of multithreaded software with asynchronous communication. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 348–359. Springer, Heidelberg (2005)
7. Carayol, A., Wohrle, S.: The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In: Proceedings of the International Conference on Foundations of Software Technology and Theoretical Computer Science, pp. 112–123 (2003)

8. Chadha, R., Viswanthan, M.: Decidability results for well-structured transition systems with auxiliary storage. Technical Report UIUCDCS-R-2007-2865, Univ. of Illiniois at Urbana-Champaign (2007)
9. Emmi, M., Majumdar, R.: Decision problems for the verification of real-time software. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC 2006. LNCS, vol. 3927, pp. 200–211. Springer, Heidelberg (2006)
10. Esparza, J., Etessami, K.: Verifying probabilistic procedural programs. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 16–31. Springer, Heidelberg (2004)
11. Finkel, A., Schnoebelen, Ph.: Well-structured transition systems everywhere! Theoretical Computer Science 256(1–2), 63–92 (2001)
12. Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D.: The nesC language: A holistic approach to networked embedded systems. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 1–11. ACM Press, New York (2003)
13. Graf, S., Saidi, H.: Construction of abstract state graphs with pvs. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
14. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. In: Proceedings of the International Conference on Architectural support for Programming Languages and Operating Systems, pp. 93–104 (2000)
15. Holub, A.: Taming Java Threads. APress (2000)
16. Jhala, R., Majumdar, R.: Interprocedural analysis of asynchronous programs. In: Proceedings of the ACM Symposium on Principles of Programming Languages, pp. 339–350. ACM Press, New York (2007)
17. Kruskal, J.B.: The theory of well-quasi-ordering: A frequently discovered concept. Journal of Combinatorial Theory: Series A 13(3), 297–305 (1972)
18. Libasync. http://pdos.csail.mit.edu/6.824-2004/async/
19. Libevent. http://www.monkey.org/provos/libevent/
20. Majumdar, R.: Personal communication
21. Mayr, R.: Decidability and Complexity of Model Checking Problems for Infinite-State Systems. PhD thesis, Technical University Munich (1998)
22. Moller, F.: Infinite results. In: Proceedings of the Conference on Concurrency Theory, pp. 195–216 (1996)
23. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Proceedings of the International Conference on Tools and Algorithms for Construction and Analysis of Systems, pp. 93–107 (2005)
24. Sen, K., Viswanathan, M.: Model checking multithreaded programs with asynchronous atomic methods. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 300–314. Springer, Heidelberg (2006)
25. Vardhan, A.: Learning to Verify Systems. PhD thesis, University of Illinois, Urbana-Champaign (2005)

A Nice Labelling for Tree-Like Event Structures of Degree 3*

Luigi Santocanale

LIF – CNRS, Université de Provence
luigi.santocanale@lif.univ-mrs.fr

Abstract. We address the problem of finding nice labellings for event structures of degree 3. We develop a minimum theory by which we prove that the labelling number of an event structure of degree 3 is bounded by a linear function of the height. The main theorem we present in this paper states that event structures of degree 3 whose causality order is a tree have a nice labelling with 3 colors. Finally, we exemplify how to use this theorem to construct upper bounds for the labelling number of other event structures of degree 3.

1 Introduction

Event structures, introduced in [1], are nowadays a widely recognized model of true concurrent computation and have found many uses since then. They are an intermediate abstract model that make it possible to relate more concrete models such as Petri Nets or higher dimensional automata [2]. They provide formal semantics of process calculi [3,4]. More recently, logicians became interested in event structures with the aim of constructing models of proof systems that are invariant under the equalities induced by the cut elimination procedure [5,6].

Our interest for event structures stems from the fact that they combine distinct approaches to the modeling of concurrent computation. On one side, language theorists have developed the theory of partially commutative monoids [7] as the basic language to approach concurrency. On the other hand, the framework of domain theory and, ultimately, order theoretic ideas have often been proposed as the proper tools to handle concurrency, see for example [8]. In this paper we pursue a combinatorial problem that lies at the intersection of these two approaches. It is the problem of finding nice labellings for event structures of fixed degree. To our knowledge, this problem has not been investigated any longer since it was posed in [9,10] and partially solved in [11].

Let us recall that an event structure is made up of a set of local events E which is ordered by a causality relation $\leq$. Moreover, a concurrency relation $\frown$, that may only relate causally independent events, is given. A global state of the computation is modeled as a clique of the concurrency relation. Global states may be organized into a poset, the coherent domain of an event structure, which represents all the concurrent non-deterministic executions of a system. Roughly

* Research supported by the *Agence Nationale de la Recherche*, project SOAPDC.

speaking, the nice labelling problem consists in representing the coherent domain of an event structure as a poset of traces or, more precisely, pomsets. That is, such a domain should be reconstructed using the standard ingredients of trace theory: an alphabet Σ, a local independence relation I, and a prefix closed subset of the free monoid L, see [12,13]. By the general theory relating traces to ordered sets, the problem always has a solution (Σ, I, L). We are asked to find a solution with the cardinality of the alphabet Σ minimal. The problem is actually equivalent to a graph coloring problem in that we can associate to an event structure a graph, of which we are asked to compute the chromatic number. The degree of an event structure is the maximal number of upper covers of some elements in the associated domain. Under the graph theoretic translation of the problem, the degree coincides with the clique number, and therefore it is a lower bound for the cardinality of a solution. A main contribution in [11] was to prove that event structures of degree 2 have a nice labelling with 2 letters, i.e. they have a solution (Σ, I, L) with $\mathrm{card}(\Sigma) = 2$. On the other hand, it was proved there that event structures of higher degrees may require more letters than the degree.

The labelling problem may be thought to be a generalization of the problem of covering a poset by disjoint chains. Dilworth's Theorem [14] states that the minimal cardinality of such a cover equals the maximal cardinality of an antichain. This theorem and the results of [11] constitute the few knowledge on the problem presently available to us. For example, we cannot state that there is some fixed $k > n$ for which every event structure of degree n has a nice labelling with at most k letters. In light of standard graph theoretic results [15], the above statement should not be taken for granted.

We present here our first results on the nice labelling problem for event structures of degree 3. We develop a minimum theory that shows that the graph of a degree 3 event structure, when restricted to an antichain, is almost acyclic and can be colored with 3 letters. This observation allows to construct an upper bound to the labelling number of such event structure as a linear function of its height. We prove then our main theorem stating that event structures of degree 3, whose causality order is a tree, have a nice labelling with 3 letters. Let us just say that such an event structure may represent a concurrent system where some processes are only allowed to fork or to take local nondeterministic choices. Finally, we exemplify how to use this theorem to construct upper bounds for the labelling number of other event structures of degree 3. In some simple cases, we obtain constant upper bounds to the labelling number, i.e. upper bounds that are functions of no parameter.

While these results do not answer the general problem, that of computing the labelling number of degree 3 event structures, we are aware that graph coloring problems may be difficult to answer. Thus we decided to present these results and share the knowledge so far acquired and also to encourage other researchers to pursue the problem. Let us mention why we believe that this and other problems in the combinatorics of concurrency deserve to be deeply investigated. The theory of event structures is now being applied within verification. A model checker, POEM, presently developed in Marseilles, makes explicit use of trace theory and of the theory of partial orders to represent the state space of a concurrent system [16]. The combinatorics of posets is exploited there and elsewhere to achieve an the efficient exploration of the global states of concurrent systems [17,18]. Thus, having a solid theoretical understanding of such combinatorics is, for us, a prerequisite and a complement for designing efficient algorithms for these kind of tools.

The paper is structured as follows. After recalling the order theoretic concepts we shall use, we introduce event structures and the nice labelling problem in section 2. In section 3 we develop the first properties of event structures of degree 3. As a result, we devise an upper bound for the labelling number of such event structures as a linear function of the height. In section 4 we present our main result stating that event structures whose underlying order is a tree may be labeled with 3 colors. In section 5 we develop a general approach to construct upper bounds to the labelling number of event structures of degree 3. Using this approach and the results of the previous section, we compute a constant upper bound for a class of degree 3 event structures that have some simplifying properties and which are consequently called simple.

Order Theoretic Preliminaries. We shall introduce event structures in the next section. For the moment being let us anticipate that part of an event structure is a set E of events which is partially ordered by a causality relation $\leq$. In this paper we shall heavily make use of order theoretic concepts. We introduce them here together with the notation that shall be used. All these concepts will apply to the poset $\langle E, \leq \rangle$ of an event structure.

A finite poset is a pair $\langle P, \leq \rangle$ where P is a finite set and $\leq$ is a reflexive, transitive and antisymmetric relation on P. A subset $X \subseteq P$ is a *lower set* if $y \leq x \in X$ implies $y \in X$. If $Y \subseteq P$, then we denote by $\downarrow Y$ the least lower set containing Y. Explicitly, $\downarrow Y = \{\, x \in P \,|\, \exists y \in Y \text{ s.t. } x \leq y \,\}$. Two elements $x, y \in P$ are *comparable* if and only if either $x \leq y$ or $y \leq x$. We write $x \simeq y$ to mean that x, y are comparable. A *chain* is sequence $x_0, \ldots, x_n$ of elements of P such that $x_0 < x_1 < \ldots < x_n$. The integer n is the length of the chain. The *height* of an element $x \in P$, noted $\mathrm{h}(x)$, is the length of the longest chain in $\downarrow \{\, x \,\}$. The height of P is $\max\{\, \mathrm{h}(x) \,|\, x \in P \,\}$. An *antichain* is a subset $X \subseteq P$ such that $x \not\simeq y$ for each pair of distinct $x, y \in X$. The *width* of $\langle P, \leq \rangle$, noted $\mathrm{w}(P, \leq)$, is the integer $\max\{\, \mathrm{card}(A) \,|\, A \text{ is an antichain} \,\}$. If the interval $\{\, z \in P \,|\, x \leq z \leq y \,\}$ is the two elements set $\{\, x, y \,\}$, then we say that x is a *lower cover* of y or that y is *an upper cover* of x. We denote this relation by $x \prec y$. The Hasse diagram of $\langle P, \leq \rangle$ is the directed graph $\langle P, \prec \rangle$. For $x \in P$, the *degree* of x, noted $\deg(x)$, is the number of upper covers of x. That is, the degree of x is the outdegree of x in the Hasse diagram. The degree of $\langle P, \leq \rangle$, noted $\deg(P, \leq)$, is the integer $\max\{\, \deg(x) \,|\, x \in P \,\}$. We shall denote by $\mathrm{f}(x)$ the number of lower covers of x (i.e. the indegree of x in the Hasse diagram). The poset $\langle P, \leq \rangle$ is *graded* if $x \prec y$ implies $\mathrm{h}(y) = \mathrm{h}(x) + 1$.

2 Event Structures and the Nice Labelling Problem

Event structures are a basic model of concurrency introduced in [1]. The definition we present here is from [2].

Definition 1. *An* event structure *is a triple* $\mathcal{E} = \langle E, \leq, \mathcal{C} \rangle$ *such that*

- $\langle E, \leq \rangle$ *is a poset, such that for each* $x \in E$ *the lower set* $\downarrow \{ x \}$ *is finite,*
- $\mathcal{C}$ *is a collection of subsets of* E *such that:*
  1. $\{ x \} \in \mathcal{C}$ *for each* $x \in E$*,*
  2. $X \subseteq Y \in \mathcal{C}$ *implies* $X \in \mathcal{C}$*,*
  3. $X \in \mathcal{C}$ *implies* $\downarrow X \in \mathcal{C}$*.*

The order $\leq$ of an event structure $\mathcal{E}$ is known as the *causality* relation between events. The collection $\mathcal{C}$ is known as the set of configurations of $\mathcal{E}$. A configuration $X \in \mathcal{C}$ of causally unrelated events – that is, an antichain w.r.t. $\leq$ – is a sort of snapshot of the global state of some distributed computation. A snapshot X may be transformed into a description of the computation that takes into account its history. This is done by adding to X the events that causally have determined events in X. That is, the history aware description is the lower set $\downarrow X$ generated by X.

Two elements $x, y \in E$ are said to be *concurrent* if $x \not\simeq y$ and there exists $X \in \mathcal{C}$ such that $x, y \in X$. Two concurrent elements will be thereby noted by $x \frown y$. It is useful to introduce a weakened version of the concurrency relation where we allow elements to be comparable: $x \overset{\frown}{\simeq} y$ if and only if $x \frown y$ or $x \simeq y$. Equivalently, $x \overset{\frown}{\simeq} y$ if and only if there exists $X \in \mathcal{C}$ such that $x, y \in X$. In many concrete models the set of configurations is completely determined by the concurrency relation.

Definition 2. *An event structure* $\mathcal{E}$ *is* coherent *if* $\mathcal{C}$ *is the set of cliques of the weak concurrency relation:* $X \in \mathcal{C}$ *if and only if* $x \overset{\frown}{\simeq} y$ *for every pair of elements* $x, y \in X$*.*

Coherent event structures are also known as event structures with binary conflict. To understand the naming let us explicitely introduce the conflict relation and two other derived relations:

- *Conflict*: $x \smile y$ if and only if $x \not\simeq y$ and $x \not\frown y$.
- *Minimal conflict*: $x \mathrel{\underline{\smile}} y$ if and only (i) $x \smile y$, (ii) $x' < x$ implies $x' \overset{\frown}{\simeq} y$, and (iii) $y' < y$ implies $x \overset{\frown}{\simeq} y'$.
- *Orthogonality*: $x \mathrel{\underline{\frown}} y$ if and only if $x \mathrel{\underline{\smile}} y$ or $x \frown y$.

A coherent event structure is completely described by a triple $\langle E, \leq, \frown \rangle$ where the latter is a symmetric relation subject to the following conditions: (i) $x \frown y$ implies $x \not\simeq y$, (ii) $x \frown y$ and $z \leq x$ implies $z \frown y$ or $z \leq y$. Similarly, a coherent event structure is completely determined by the order and the conflict relation. In this paper we shall deal with coherent event structures only and, from now on, event structure will be a synonym for coherent event structure.

We shall focus mainly on the orthogonality relation. The rest of this section will explain the role played by this relation. Let us observe now that two orthogonal elements are called independent in [11]. We prefer however not to use this terminology: we shall frequently make use of standard graph theoretic language and argue about cliques, not on their dual, independent sets. The orthogonality relation clearly is symmetric and moreover it inherits from the concurrency relation the following property: if $x \mathrel{\overline{\frown}} y$ and $z \leq x$, then $z \mathrel{\overline{\frown}} y$ or $z \leq y$.

Definition 3. *A* nice labelling *of an event structure $\mathcal{E}$ is a pair (λ, Σ), where Σ is a finite alphabet and $\lambda : E \longrightarrow \Sigma$ is such that $\lambda(x) \neq \lambda(y)$ whenever $x \mathrel{\overline{\frown}} y$.*

That is, if we let $\mathcal{G}(\mathcal{E})$ – the graph of $\mathcal{E}$ – be the pair $\langle E, \overline{\frown} \rangle$, a labelling of $\mathcal{E}$ is a coloring of the graph $\mathcal{G}(\mathcal{E})$. For a graph G, let $\gamma(G)$ denote its chromatic number. Let us say then that the *labelling number* of $\mathcal{E}$ is $\gamma(\mathcal{G}(\mathcal{E}))$. The *nice labelling problem* for a class $\mathcal{K}$ of event structures amounts to computing the number $\gamma(\mathcal{K}) = \max\{\, \gamma(\mathcal{G}(\mathcal{E})) \,|\, \mathcal{E} \in \mathcal{K} \,\}$. To understand the origins of this problem, let us recall the definition of the domain of an event structure.

Definition 4. *The domain $\mathcal{D}(\mathcal{E})$ of an event structure $\mathcal{E} = \langle E, \leq, \mathcal{C} \rangle$ is the collection of lower sets in $\mathcal{C}$, ordered by subset inclusion.*

Following a standard axiomatization in theoretical computer science [2] $\mathcal{D}(\mathcal{E})$ is a stable domain which is coherent if $\mathcal{E}$ is coherent. Stable means that $\mathcal{D}(\mathcal{E})$ is essentially a distributive lattice. As a matter of fact, if $\mathcal{E}$ is finite, then a possible alternative axiomatization of the poset $\mathcal{D}(\mathcal{E})$ is as follows. It is easily seen that the collection $\mathcal{D}(\mathcal{E})$ is closed under binary intersections, hence it is a finite meet semilattice without a top element, a chopped lattice in the sense of [19, Chapter 4]. Also the chopped lattice is distributive, meaning that whenever $X, Y, Z \in \mathcal{D}(\mathcal{E})$ and $X \cup Y \in \mathcal{D}(\mathcal{E})$, then $Z \cap (X \cup Y) = (Z \cap X) \cup (Z \cap Y)$. It can be shown that every distributive chopped lattice is isomorphic to the domain of a finite – not necessarily coherent – event structure.

Lemma 1. *A set $\{\, x_1, \ldots, x_n \,\}$ is a clique in the graph $\mathcal{G}(\mathcal{E})$ iff there exists $I \in \mathcal{D}(\mathcal{E})$ such that $I \cup \{\, x_i \,\}$, $i = 1, \ldots, n$, are distinct upper covers of I in the domain $\mathcal{D}(\mathcal{E})$.*

The Lemma shows that a nice labelling $\lambda : E \longrightarrow \Sigma$ allows to label the edges of the Hasse diagram of $\mathcal{D}(\mathcal{E})$ so that: (i) outgoing edges from the same source vertex have distinct labels, (ii) perspective edges – i.e. edges $I_0 \prec I_1$ and $J_0 \prec J_1$ such that $I_0 = I_1 \cap J_0$ and $J_1 = I_1 \cup J_0$ – have the same label. These conditions are necessary and sufficient to show that $\mathcal{D}(\mathcal{E})$ is order isomorphic to a consistent (but not complete) set of P-traces (or pomsets) on the alphabet Σ in the sense of [12].

The *degree* of an event structure $\mathcal{E}$ is the degree of the domain $\mathcal{D}(\mathcal{E})$, that is, the maximum number of upper covers of a lower set I within the poset $\mathcal{D}(\mathcal{E})$. Lemma 1 shows that the degree of $\mathcal{E}$ is equal to the size of a maximal clique in $\mathcal{G}(\mathcal{E})$, i.e. to the clique number of $\mathcal{G}(\mathcal{E})$. Henceforth, the degree of $\mathcal{E}$ is a lower bound to $\gamma(\mathcal{G}(\mathcal{E}))$. The following Theorems state the few results on the nice labelling problem that are available in the literature.

Theorem 1 (see [14]). *Let $\mathcal{NC}_n$ be the class of event structures of degree at most n with empty conflict relation. Then $\gamma(\mathcal{NC}_n) = n$.*

Theorem 2 (see [11]). *Let $\mathcal{K}_n$ be the class of event structures of degree at most n. Then $\gamma(\mathcal{K}_n) = n$ if $n \leq 2$ and $\gamma(\mathcal{K}_n) \geq n + 1$ otherwise.*

The last theorem has been our starting point for investigating the nice labelling problem for event structures of degree 3.

3 Cycles and Antichains

From now on, in this and the following sections, $\mathcal{E} = \langle E, \leq, \mathcal{C} \rangle$ will denote a coherent event structure of degree at most 3. We begin our investigation of the nice labelling problem by studying the restriction to an antichain of the graph $\mathcal{G}(\mathcal{E})$. The main tool we shall use is the following Lemma. It is a straightforward generalization of [11, Lemma 2.2] to degree 3. In [20] we proposed generalizations of this Lemma to higher degrees, pointing out their strong geometrical flavor.

Lemma 2. *Let $\{ x_0, x_1, x_2 \}, \{ x_1, x_2, x_3 \}$ be two size 3 cliques in the graph $\mathcal{G}(\mathcal{E})$ sharing the same face $\{ x_1, x_2 \}$. Then x_0, x_3 are comparable.*

Proof. Let us suppose that x_0, x_3 are not comparable. It is not possible that $x_0 \mathrel{\overline{\frown}} x_3$, since then we have a size 4 clique in the graph $\mathcal{G}(\mathcal{E})$. Thus $x_0 \smile x_3$ and we can find $x_0' \leq x_0$ and $x_3' \leq x_3$ such that $x_0' \mathrel{\underline{\smile}} x_3'$. We claim that $\{ x_0', x_1, x_2, x_3' \}$ is a size 4 clique in $\mathcal{G}(\mathcal{E})$, thus reaching a contradiction.

If $x_0' \not\mathrel{\overline{\frown}} x_1$, then $x_0' \leq x_1$, but this, together with $x_1 \mathrel{\overline{\frown}} x_3$, contradicts $x_0' \mathrel{\underline{\smile}} x_3'$. Similalry, $x_0' \mathrel{\overline{\frown}} x_2$, $x_3' \mathrel{\overline{\frown}} x_1$, $x_3' \mathrel{\overline{\frown}} x_2$. □

We are going to improve on the previous Lemma. To this goal, let us say that a sequence $x_0 x_1 \ldots x_{n-1} x_n$ is a *straight cycle* if $x_n = x_0$, $x_i \mathrel{\overline{\frown}} x_{i+1}$ for $i = 0, \ldots, n-1$, $x_i \not\asymp x_j$ whenever $i, j \in \{ 0, \ldots, n-1 \}$ and $i \neq j$. As usual, the integer n is the length of the cycle. Observe that a straight cycle is simple, i.e., a part from the endpoints of the cycle, it does not visit twice the same vertex . The height of a straight cycle $C = x_0 x_1 \ldots x_n$ is the integer

$$\mathrm{h}^{+}(C) = \sum_{i=0,\ldots,n-1} \mathrm{h}(x_i) + 1 \,.$$

The definition of h^{+} implies that if C' is a subcycle of C induced by a chord, then $\mathrm{h}^{+}(C') < \mathrm{h}^{+}(C)$.

Proposition 1. *The graph $\mathcal{G}(\mathcal{E})$ does not contain a straight cycle of length strictly greater than 3.*

Proof. Let $\mathcal{SC}_{\geq 4}$ be the collection of straight cycles in $\mathcal{G}(\mathcal{E})$ whose length is at least 4. We shall show that if $C \in \mathcal{SC}_{\geq 4}$, then there exists $C' \in \mathcal{SC}_{\geq 4}$ such that $\mathrm{h}^{+}(C') < \mathrm{h}^{+}(C)$.

Let C be the straight cycle $x_0 \overline{\frown} x_1 \overline{\frown} x_2 \ldots x_{n-1} \overline{\frown} x_n = x_0$ where $n \geq 4$. Let us suppose that this cycle has a chord. It follows, by Lemma 2, that $n > 4$. Hence the chord divides the cycle into two straight cycles, one of which has still length at least 4 and whose height is less than the height of C, since it contains a smaller number of vertices.

Otherwise C has no chord and $x_0 \not\overline{\frown} x_2$. This means that either there exists $x_0' < x_0$ such that $x_0' \not\frown x_0$, or there exists $x_2' < x_2$ such that $x_0 \not\frown x_2'$. By symmetry, we can assume the first case holds. As in the proof of Lemma 2 $\{ x_0', x_1, x_2, x_3 \}$ form an antichain, and $x_0' x_1 x_2 x_3$ is a path. Let C' be the set $\{ x_0' x_1, \ldots x_{n-1} x_0' \}$. If C' is an antichain, then C' is a straight cycle such that $\mathrm{h}^+(C') < \mathrm{h}^+(C)$. Otherwise the set $\{ j \in \{ 4, \ldots, n-1 \} \,|\, x_j \geq x_0' \}$ is not empty. Let i be the minimum in this set, and observe that $x_{i-1} \overline{\frown} x_i$ and $x_0' \leq x_i$ but $x_0' \not\leq x_{i-1}$ implies $x_{i-1} \overline{\frown} x_0'$. Thus $\tilde{C} = x_0' x_1 x_2 x_3 \ldots x_{i-1} x_0'$ is a straight cycle of lenght at least 4 such that $\mathrm{h}^+(\tilde{C}) < \mathrm{h}^+(C)$. □

Corollary 1. *Any subgraph of $\mathcal{G}(\mathcal{E})$ induced by an antichain can be colored with 3 colors.*

Proof. Since the only cycles have length at most 3, such an induced graph is chordal and its clique number is 3. It is well known that the chromatic number of chordal graphs equals their clique number [21]. □

In the rest of this section we exploit the previous observations to construct upper bounds for the labelling number of $\mathcal{E}$. We remark that these upper bounds might appear either too abstract, or too trivial. On the other hand, we believe that they well illustrate the kind of problems that arise when trying to build complex event structures that might have labelling number greater than 4.

A *stratifying function* for $\mathcal{E}$ is a function $h : E \longrightarrow \mathbb{N}$ such that, for each $n \geq 0$, the set $\{ x \in E \,|\, h(x) = n \}$ is an antichain. The height function is a stratifying function. Also $\varsigma(x) = \mathrm{card}\{ y \in E \,|\, y < x \}$ is a stratifying function. With respect to a stratifying function h the h-skewness of $\mathcal{E}$ is defined by

$$\mathrm{skew}_h(\mathcal{E}) = \max\{ |h(x) - h(y)| \,|\, x \overline{\frown} y \} .$$

More generally, the skewness of $\mathcal{E}$ is defined by

$$\mathrm{skew}(\mathcal{E}) = \min\{ \mathrm{skew}_h(\mathcal{E}) \,|\, h \text{ is a stratifying function } \} .$$

Proposition 2. *If* $\mathrm{skew}(\mathcal{E}) < n$ *then* $\gamma(\mathcal{G}(\mathcal{E})) \leq 3n$.

Proof. Let h be a stratifying function such that $|h(x) - h(y)| < n$ whenever $x \overline{\frown} y$. For each $k \geq 0$, let $\lambda_k : \{ x \in E \,|\, h(x) = k \} \longrightarrow \{ a, b, c \}$ be a coloring of the graph induced by $\{ x \in E \,|\, h(x) = k \}$. Define $\lambda : E \longrightarrow \{ a, b, c \} \times \{ 0, \ldots, n-1 \}$ as follows:

$$\lambda(x) = (\lambda_{h(x)}(x), h(x) \bmod n) .$$

Let us suppose that $x \overline{\frown} y$ and $h(x) \geq h(y)$, so that $0 \leq h(x) - h(y) < n$. If $h(x) = h(y)$, then by construction $\lambda_{h(x)}(x) = \lambda_{h(y)}(x) \neq \lambda_{h(y)}(y)$. Otherwise

$h(x) > h(y)$ and $0 \leq h(x) - h(y) < n$ implies $h(x) \bmod n \neq h(y) \bmod n$. In both cases we obtain $\lambda(x) \neq \lambda(y)$. □

An immediate consequence of Proposition 2 is the following upper bound for the labelling number of $\mathcal{E}$:

$$\gamma(\mathcal{G}(\mathcal{E})) \leq 3(\mathrm{h}(\mathcal{E}) + 1)\,.$$

To appreciate the upper bound, consider that another approximation to the labelling number of $\mathcal{E}$ is provided by Dilworth's Theorem [14], stating that $\gamma(\mathcal{G}(\mathcal{E})) \leq \mathrm{w}(\mathcal{E})$. To compare the two bounds, consider that there exist event structures of degree 3 whose width is an exponential function of the height.

4 An Optimal Nice Labelling for Trees and Forests

We prove in this section the main contribution of this paper. Assuming that $\langle E, \leq \rangle$ is a tree or a forest, then we define a labelling with 3 colors, and prove it is a nice labelling. Since clearly we can construct a tree which needs at least three colors, such a labelling is optimal. Before defining the labelling, we shall develop a small amount of observations.

Definition 5. *We say that two distinct events are* twins *if they have the same set of lower covers.*

Clearly if x, y are twins, then $z < x$ if and only if $z < y$. More importantly, if x, y are twins, then the relation $x \mathrel{\overline{\frown}} y$ holds. As a matter of fact, if $x' < x$ then $x' < y$, hence $x' \mathrel{\underline{\frown}} y$. Similarly, if $y' < y$ then $y' \mathrel{\underline{\frown}} x$. It follows that a set of events having the same lower covers form a clique in $\mathcal{G}(\mathcal{E})$, hence it has at most the degree of an event structure, 3 in the present case. To introduce the next Lemmas, if $x \in Y \subseteq E$, define

$$O_x^Y = \{\, z \mid z \mathrel{\overline{\frown}} x \text{ and } y \not\leq z, \text{ forall } y \in Y \,\}\,.$$

If $Y = \{\, x, y \,\}$, then we shall abuse of notation and write $O_x^{x,y}$, $O_x^{y,x}$ as synonyms of O_x^Y. Thus $z \in O_x^{x,y}$ if and only if $z \mathrel{\overline{\frown}} x$ and $y \not\leq z$.

Lemma 3. *If x, y, z are parwise distinct twins, then $O_x^{\{x,y,z\}} = \emptyset$.*

Proof. Let us suppose that $w \in O_x^{\{x,y,z\}}$. If $w \mathrel{\overline{\frown}} y$, then $w \simeq z$ by Lemma 2. Since $z \not\leq w$, then $w < z$. However this implies $w < x$, contradicting $w \mathrel{\overline{\frown}} x$. Hence $w \mathrel{\not\overline{\frown}} y$ and we can find $w' \leq w$, $y' \leq y$ such that $w' \mathrel{\underline{\smile}} y'$. It cannot be the case that $y' < y$, otherwise $y' < x$ and the pair (w', y'), properly covered by the pair (w, x), cannot be a minimal conflict. Thus $w' < w$, and y' equals to y. We claim that $w' \in O^{\{x,y,z\}}$. As a matter of fact, w' cannot be above any of the elements in $\{\, x, y, z \,\}$, otherwise w would have the same property. From $w \mathrel{\overline{\frown}} x$ and $w' < w$, we deduce that $w' \mathrel{\overline{\frown}} x$ or $w' \leq x$. If the latter, then $w' < x$, so that $w' < y$, contradicting $w' \mathrel{\underline{\smile}} y$. Therefore $w' \mathrel{\overline{\frown}} x$ and $\{\, w', x, y \,\}, \{\, x, y, z \,\}$ are two 3-cliques sharing the same face $\{\, x, y \,\}$. As before, $w' \simeq z$, leading to a contradiction. □

Lemma 4. *If x, y are twins, then $O_x^{x,y}, O_y^{x,y}$ are comparable w.r.t. set inclusion and $O_x^{x,y} \cap O_y^{x,y}$ is a linear order.*

Proof. We observe first that if $z \in O_x^{x,y}$ and $w \in O_y^{x,y}$ then $z \simeq w$. As a consequence $O_x^{x,y} \cap O_y^{x,y}$ is linearly ordered.

Let us suppose that there exists $z \in O_x^{x,y}$ and $w \in O_y^{x,y}$ such that $z \not\simeq w$. Observe then that $\{\, z, x, y, w \,\}$ is an antichain: $y \not\leq z$, and $z < y$ implies $z < x$, which is not the case due to $z \mathrel{\underline{\frown}} x$. Thus $z \not\simeq y$ and similarly $w \not\simeq x$.

Since there cannot be a length 4 straight cycle, we deduce $z \mathrel{\not\frown} w$. Let $z' \leq z$ and $w' \leq w$ be such that $z' \mathrel{\underline{\smile}} w'$. We claim first that $z' \mathrel{\underline{\frown}} x$. Otherwise, $z' \leq x$ and $z' < x$, since $z' = x$ implies $x \leq z$. The relation $z' < x$ in turn implies $z' < y$, which contradicts $z' \mathrel{\underline{\smile}} w'$. Also it cannot be the case that $y \leq z'$, since otherwise $y \leq z$. Thus, we have argued that $z' \in O_x^{x,y}$. Similarly $w' \in O_y^{x,y}$. As before $\{\, z', x, y, w' \,\}$ is an antichain, hence z', x, y, w' also form a length 4 straight cycle, a contradiction.

Observe now that $w \leq z \in O_x^{x,y}$ and $w \not\leq x$ implies $w \in O_x^{x,y}$. From $w \leq z \mathrel{\underline{\frown}} x$ deduce $w \mathrel{\underline{\frown}} x$ or $w \leq x$. Since $w \not\leq x$, then $w \mathrel{\underline{\frown}} x$. Also, if $y \leq w$ then $y \leq z$, which is not the case.

Let $z \in O_x^{x,y} \setminus O_y^{x,y}$, pick any $w \in O_y^{x,y}$ and recall that z, w are comparable. We cannot have $z \leq w$ since $z \not\leq y$ implies then $z \in O_y^{x,y}$. Hence $w < z \in O_x^{x,y}$ and $w \not\leq x$ imply $w \in O_x^{x,y}$ by the previous observation. □

The following Lemma will prove to be the key observation in defining later a nice labelling.

Lemma 5. *Let (x, y) (z, w) be two pairs of pairwise distinct twins such that $z \in O_x^{x,y} \cap O_y^{x,y}$ and $w \not\leq x$. Then $O_z^{w,z} \supset O_w^{w,z}$.*

Proof. If $O_z^{w,z} \not\supset O_w^{w,z}$, then $O_z^{w,z} \subseteq O_w^{w,z}$ by Lemma 4. Since $w \not\leq x$ and $w \neq y$, then $w \not\leq y$. We have shown that $x, y \in O_z^{w,z}$, hence $x, y \in O_w^{w,z}$. It follows that $\{\, x, y, z, w \,\}$ is a size 4 clique, a contradiction. □

We come now to discuss some subsets of E for which we shall prove that there exists a nice labelling with 3 letters. The intuitive reason for that is the presence of many twins.

Definition 6. *A subset $T \subseteq E$ is a* tree *if and only if*

- *each $x \in T$ has exactly one lower cover $\pi(x) \in E$,*
- *T is convex: $x, z \in T$ and $x < y < z$ implies $y \in T$,*
- *if x, y are minimal in T, then $\pi(x) = \pi(y)$.*

If T is a tree and $x \in T$, the height of x in T, noted $\mathrm{h}_T(x)$, is the cardinality of the set $\{\, y \in T \mid y < x \,\}$. A linear ordering $\lhd$ on T is said to be compatible with the height if it satisfies

$$\mathrm{h}_T(x) < \mathrm{h}_T(y) \text{ implies } x \lhd y\,. \qquad \text{(HEIGHT)}$$

It is not difficult to see that such a linear ordering always exists. With respect to such linear ordering, define

$$O_{\lhd}(x) = \{\, y \in T \mid y \mathrel{\overline{\frown}} x \text{ and } y \lhd x \,\}\,, \quad x \in T.$$

We shall represent $O_{\lhd}(x)$ as the disjoint union of $C_{\lhd}(x)$ and $L_{\lhd}(x)$ where

$$C_{\lhd}(x) = \{\, y \in O_{\lhd}(x) \mid z \prec x \text{ implies } z \leq y \,\}\,, \qquad L_{\lhd}(x) = O_{\lhd}(x) \setminus C_{\lhd}(x)\,.$$

With respect these sets $C_{\lhd}(x), L_{\lhd}(x)$, $x \in T$, we develop a series of observations.

Lemma 6. *If $y \in C_{\lhd}(x)$ then x, y are twins. Consequently there can be at most two elements in $C_{\lhd}(x)$.*

Proof. If $y \in C_{\lhd}(x)$, then $y \lhd x$ and $\mathrm{h}_T(y) \leq \mathrm{h}_T(x)$. Since y is above any lower cover of x, and distinct from such a lower cover, then $\mathrm{h}_T(x) \leq \mathrm{h}_T(y)$. It follows that $\mathrm{h}_T(x) = \mathrm{h}_T(y)$, hence if z is a lower cover of x, then it is also a lower cover of y. Since x, y have exactly one lower cover, it follows that x, y are twins. □

Lemma 7. *If x, y are twins, then $L_{\lhd}(x) \subseteq O_x^{x,y}$. If $z \in L_{\lhd}(x)$ and $z' \in O_x^{x,y}$ is such that $z' \leq z$, then $z' \in L_{\lhd}(x)$. That is, $L_{\lhd}(x)$ is a lower set of $O_x^{x,y}$.*

Proof. Let $z \in L_{\lhd}(x)$, so that $z \mathrel{\overline{\frown}} x$ and $z \mathrel{\overline{\frown}} \pi(x)$. The relation $y \leq z$ implies that $\pi(x) = \pi(y) \leq y \leq z$, and hence contradicts $z \mathrel{\overline{\frown}} \pi(x)$. Hence $y \not\leq z$ and $z \in O_x^{x,y}$. Let us suppose that $z' < z$ and $z' \mathrel{\overline{\frown}} x$. Then $\mathrm{h}(z') < \mathrm{h}(z)$ and $z' \lhd z$, so that $z' \in C_{\lhd}(x)$. Since $z' \mathrel{\overline{\frown}} x$ then either $z' \mathrel{\overline{\frown}} \pi(x)$, or $\pi(x) \leq z'$. However, the latter property implies $\pi(x) \leq z$, which is not the case. Therefore $z' \mathrel{\overline{\frown}} \pi(x)$ and $z' \in L_{\lhd}(x)$. □

Lemma 8. *If x, y, z are pairwise distinct twins, then $L_{\lhd}(x) = \emptyset$ and z is the minimal element of $O_x^{x,y} \cap O_y^{x,y}$. In particular $O_{\lhd}(x) = C_{\lhd}(x) \subseteq \{\, y, z \,\}$.*

Proof. By the previous observation $L_{\lhd}(x) \subseteq O_x^{x,y}$ and, similarly, $L_{\lhd}(x) \subseteq O_x^{x,z}$. Hence $L_{\lhd}(x) \subseteq O_x^{x,y} \cap O_x^{x,z} = O_x^{x,y,z} = \emptyset$, by Lemma 3. Since $x \mathrel{\overline{\frown}} z$ and $y \mathrel{\overline{\frown}} z$ then $z \in O_x^{x,y} \cap O_y^{x,y}$. If $z' < z$ then $z' < x$ and $z' < y$ hence $z' \notin O_x^{x,y} \cup O_y^{x,y}$.
Finally, the relation $C_{\lhd}(x) \subseteq \{\, y, z \,\}$ follows from Lemma 6. □

The previous observations motivate us to introduce the next Definition.

Definition 7. *Let us say that $x, y \in T$ are a* proper pair of twins *if they are distinct and $\{\, z \mid \pi(z) = \pi(x) \,\} = \{\, x, y \,\}$. We say that a linear order $\lhd$ on T is compatible with proper pair of twins if it satisfies* (HEIGHT) *and moreover*

$$O_x^{x,y} \supset O_y^{x,y} \text{ implies } x \lhd y\,, \qquad \text{(TWINS)}$$

for each proper pair of twins x, y.

Again is not difficult to see that such a linear order always exists and in the following we shall assume that $\lhd$ satisfies both (HEIGHT) and (TWINS).

We are ready to define a partial labelling λ of the event structure $\mathcal{E}$. The function λ will have T as its domain. W.r.t. $\lhd$ let us say that $x \in T$ is *principal* if $C_{\lhd}(x) = \emptyset$. Let $\Sigma = \{\, a_0, a_1, a_2 \,\}$ be a three elements totally ordered alphabet. The labelling $\lambda : T \longrightarrow \Sigma$ is defined by induction on $\lhd$ as follows:

1. If $x \in T$ is principal and $\mathrm{h}_T(x) = 0$, then we let $\lambda(x) = a_0$.
2. If $x \in T$ is principal and $\mathrm{h}_T(x) \geq 1$, let $\pi(x)$ be its unique lower cover. Since $\pi(x) \in T$ and $\pi(x) \lhd x$, $\lambda(\pi(x))$ is defined and we let $\lambda(x) = \lambda(\pi(x))$.
3. If x is not principal and $L_\lhd(x) = \emptyset$, then, by Lemma 6, we let $\lambda(x)$ be the least symbol not in $\lambda(C_\lhd(x))$.
4. If x is not principal and $L_\lhd(x) \neq \emptyset$ then:
   – by Lemma 8 $C_\lhd(x) = \{\, y \,\}$ is a singleton and x, y is a proper pair of twins,
   – by Lemma 7 $L_\lhd(x)$ is a lower set of $O^{x,y}_x$. By the condition (TWINS), $O^{x,y}_x \subseteq O^{x,y}_y$, so that $O^{x,y}_x$ is a linear order. Let therefore z_0 be the common least element of $L_\lhd(x)$ and $O^{x,y}_x$.

   We let $\lambda(x)$ be the unique symbol not in $\lambda(\{\, y, z_0 \,\})$.

Proposition 3. *For each $x, y \in T$, if $x \frown y$ then $\lambda(x) \neq \lambda(y)$.*

Proof. It suffices to prove that $\lambda(y) \neq \lambda(x)$ if $y \in O_\lhd(x)$. The statement is proved by induction on $\lhd$. Let us suppose the statement is true for all $z \lhd x$.

(i) If $\mathrm{h}_T(x) = 0$ then x is minimal in T, so that $O_\lhd(x) = C_\lhd(x)$. If moreover x is principal then $O_\lhd(x) = C_\lhd(x) = \emptyset$, so that the statement holds trivially.

(ii) If x is principal and $\mathrm{h}_T(x) \geq 1$, then its unique lower cover $\pi(x)$ belongs to T. Observe that $O_\lhd(x) = L_\lhd(x) = \{\, y \in T \mid y \lhd x \text{ and } y \frown \pi(x) \,\}$, so that if $y \in O_\lhd(x)$, then $y \frown \pi(x)$. Since $y \lhd x$ and $\pi(x) \lhd x$, and either $y \in O_\lhd(\pi(x))$ or $\pi(x) \in O_\lhd(y)$, it follows that $\lambda(x) = \lambda(\pi(x)) \neq \lambda(y)$ from the inductive hypothesis.

(iii) If x is not principal and $L_\lhd(x) = \emptyset$, then $O_\lhd(x) = C_\lhd(x)$ and, by construction, $\lambda(y) \neq \lambda(x)$ whenever $y \in O_\lhd(x)$.

(iv) If x is not principal and $L_\lhd(x) \neq \emptyset$, then let $C_\lhd(x) = \{\, y \,\}$ and let z_0 be the common least element of $L_\lhd(x)$ and $O^{x,y}_x$. Since by construction $\lambda(x) \neq \lambda(y)$, to prove that the statement holds for x, it is enough to pick $z \in L_\lhd(x)$ and argue that $\lambda(z) \neq \lambda(x)$. We claim that each element $z \in L_\lhd(x) \setminus \{\, z_0 \,\}$ is principal. If the claim holds, then $\lambda(z) = \lambda(\pi(z))$, so that $\lambda(z) = \lambda(z_0)$ is inductively deduced.

Suppose therefore that there exists $z \in L_\lhd(x)$ which is not principal and let $w \in C_\lhd(z)$. Observe that x, y form a proper pair of twins, since otherwise $L_\lhd(x) = \emptyset$ by Lemma 8. Similarly w, z form a proper pair of twins: otherwise, if z, w, u are pairwise distinct twins, then either $w \leq x$ or $u \leq x$ by Lemma 3. However this is not possible, since for example $z_0 \leq \pi(x) < u \leq x$ contradicts $z_0 \frown x$.

Since $y \lhd x$, condition (TWINS) implies $O^{x,y}_x \subseteq O^{x,y}_y$, and hence $z \in O^{x,y}_x \cap O^{x,y}_y$. If $w \in C_\lhd(z)$, then we cannot have $w \leq x$ or $w = y$, since again we would deduce $z_0 \leq x$. Thus Lemma 5 implies $O^{w,z}_z \supset O^{w,z}_w$. On the other hand, $w \lhd z$ and condition (TWINS) implies $O^{w,z}_z \subseteq O^{w,z}_w$.

Thus, we have reached a contradiction by assuming $C_\lhd(z) \neq \emptyset$. It follows that z is principal. □

The obvious corollary of Proposition 3 is that if $\mathcal{E}$ is already a sort of tree, then it has a nice labelling with 3 letters. We state this fact as the following Theorem, after we have made precise the meaning of the phrase "$\mathcal{E}$ is a sort of tree."

Definition 8. *Let us say that $\mathcal{E}$ is a* forest *if every element has at most one lower cover. Let $\mathcal{F}_3$ be the class of event structures of degree 3 that are forests.*

Theorem 3. *The labelling number of the class $\mathcal{F}_3$ is* 3*.*

As a matter of fact, let $\mathcal{E}$ be a forest, and consider the event structure $\mathcal{E}_\perp$ obtained from $\mathcal{E}$ by adding a new bottom element $\perp$. Remark that the graph $\mathcal{G}(\mathcal{E}_\perp)$ is the same graph as $\mathcal{G}(\mathcal{E})$ apart from the fact that an isolated vertex $\perp$ has been added. The set of events E is a tree within $\mathcal{E}_\perp$, hence the graph induced by E in $\mathcal{G}(\mathcal{E}_\perp)$ can be colored with three colors. But this graph is exactly $\mathcal{G}(\mathcal{E})$.

5 More Upper Bounds

The results presented in the previous section exemplify a remarkable property of event structures of degree 3: many types of subsets of events induce a subgraph of $\mathcal{G}(\mathcal{E})$ that can be colored with 3 colors. These include antichains by Corollary 1, trees by Proposition 3, and lower sets in $\mathcal{C}$, that is configurations of $\mathcal{E}$. As a matter fact, if $X \in \mathcal{C}$, then $\mathrm{w}(X) \leq 3$, so that such a subset can be labeled with 3 letters by Dilworth's Theorem. Also, recall that the star of an event $x \in E$ is the subgraph of $\mathcal{G}(\mathcal{E})$ induced by the subset $\{\, x \,\} \cup \{\, y \in E \mid y \overline{\frown} x \,\}$. A star can also be labeled with 3 letters. To understand the reason, let $\mathcal{N}_x$ be the event structure $\langle \{\, y \mid y \overline{\frown} x \,\}, \leq_x, \frown_x \rangle$, where $\leq_x$ and $\frown_x$ are the restrictions of the causality and concurrency relations to the set of events of $\mathcal{N}_x$.

Lemma 9. *The degree of $\mathcal{N}_x$ is strictly less than* $\deg(\mathcal{E})$*.*

Proof. The lemma follows since if $y \overline{\frown}_x z$ in $\mathcal{N}_x$, then $y \overline{\frown} z$ in $\mathcal{E}$. As a matter of fact, let us suppose that $y \overline{\frown}_x z$ in $\mathcal{N}_x$ and $y' < y$. If $y' \in \mathcal{N}_x$, then $y' \underline{\frown} z$. If $y' \notin \mathcal{N}_x$, then $y' < x$. It follows then from $z \overline{\frown} x$ and $y' < x$ that $y' \underline{\frown} x$. Similarly, if $z' < z$ then $z' \underline{\frown} x$. □

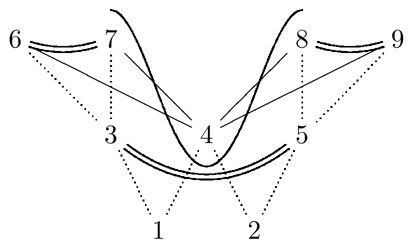
Hence, if $\deg(\mathcal{E}) = 3$, then $\deg(\mathcal{N}_x) \leq 2$, and it can be labeled with 2 letters, by [11]. It follows that star of x can be labeled with 3 letters.

It might be asked whether this property can be exploited to construct nice labellings. The positive answer comes from a standard technique in graph theory [22]. Consider a partition $\mathcal{P} = \{\, [z] \mid z \in E \,\}$ of the set of events such that each equivalence class $[z]$ has a labelling with 3 letters. Define the quotient graph $\mathcal{G}(\mathcal{P}, \mathcal{E})$ as follows: its vertexes are the equivalence classes of $\mathcal{P}$ and $[x] \overline{\frown} [y]$ if and only if there exists $x' \in [x]$, $y' \in [y]$ such that $x' \overline{\frown} y'$.

Proposition 4. *If the graph $\mathcal{G}(\mathcal{P}, \mathcal{E})$ is n-coloriable, then $\mathcal{E}$ has a labelling with $3n$ colors.*

Proof. For each equivalence class $[x]$ choose a labelling $\lambda_{[x]}$ of $[x]$ with an alphabet with 3 letters. Let λ_0 a coloring of the graph $\mathcal{G}(\mathcal{P}, \mathcal{E})$ and define $\lambda(x) = (\lambda_{[x]}(x), \lambda_0([x]))$. Then λ is a labelling of $\mathcal{E}$: if $x \overline{\frown} y$ and $[x] = [y]$, then $\lambda_{[x]}(x) = \lambda_{[y]}(x) \neq \lambda_{[y]}(y)$ and otherwise, if $[x] \neq [y]$, then $[x] \overline{\frown} [y]$ so that $\lambda_0([x]) \neq \lambda_0([y])$. □

The reader should remark that Proposition 4 generalizes Proposition 2. The Proposition also suggests that a finite upper bound for the labelling number of event structures of degree 3 might indeed exist.



We conclude the paper by exemplifying how to use the Labelling Theorem on trees and the previous Lemma to construct a finite upper bound for the labelling number of event structures that we call simple due to their additional simplifying properties. Consider the event structure on the right and name it $\mathcal{S}$. In this picture we have used dotted lines for the edges of the Hasse diagram of $\langle E, \leq \rangle$, simple lines for maximal concurrent pairs, and double lines for minimal conflicts. Concurrent pairs $x \frown y$ that are not maximal, i.e. for which there exists x', y' such that $x' \overline{\frown} y'$ and either $x < x'$ or $y < y'$, are not drawn. We leave the reader to verify that a nice labelling of $\mathcal{S}$ needs at least 4 letters. On the other hand, it shouldn't be difficult to see that a nice labelling with 4 letters exists. To obtain it, take apart events with at most 1 lower cover from the others, as suggested in the picture. Use then the results of the previous section to label with three letters the elements with at most one lower cover, and label the only element with two lower covers with a forth letter.

A formalization of this intuitive method leads to the following Definition and Proposition.

Definition 9. *We say that an event structure is* simple *if*

1. *it is graded, i.e.* $\mathrm{h}(x) = \mathrm{h}(y) - 1$ *whenever* $x \prec y$,
2. *every size* 3 *clique of* $\mathcal{G}(\mathcal{E})$ *contains a minimal conflict.*

The event structure $\mathcal{S}$ is simple and proves that even simple event structures cannot be labeled with just 3 letters.

Proposition 5. *Every simple event structure of degree* 3 *has a nice labelling with* 12 *letters.*

Proof. Recall that $\mathrm{f}(x)$ is the number of lower covers of x and let $E_n = \{\, x \in E \mid \mathrm{f}(x) = n \,\}$. Observe that a simple $\mathcal{E}$ is such that $E_3 = \emptyset$: if $x \in E_3$, then its three lower covers form a clique of concurrent events. Also, by considering the lifted event structure $\mathcal{E}_\perp$, introduced at the end of section 4, we can assume that $\mathrm{card}(E_0) = 1$, i.e. $\mathcal{E}$ has just one minimal element which necessarily is isolated in the graph $\mathcal{G}(\mathcal{E})$.

Let $\lhd$ be a linear ordering of E compatible with the height. W.r.t. this linear ordering we shall use a notation analogous to the one of the previous section: we let

$$Q_{\lhd}(x) = \{\, y \in E \,|\, y \lhd x \text{ and } y \overline{\frown} x \,\}\,, \quad C(x) = \{\, y \in E \mid y' \prec y \text{ implies } y' \prec x \,\}\,.$$

Claim. The subgraph of $\mathcal{G}(\mathcal{E})$ induced by E_2 can be colored with 3-colors.

We claim first that if $x \in E_2$ then $Q_{\lhd}(x) \subseteq C(x)$. Let $y \in Q_{\lhd}(x)$ and let x_1, x_2 be the two lower covers of x. From $x_i < x \overline{\frown} y$ it follows $x_i < y$ or $x_i \frown y$. If $x_i \frown y$ for $i = 1, 2$, then y, x_1, x_2 is a clique of concurrent events. Therefore, at least one lower cover of x is below y, let us say $x_1 < y$. It follows that $\mathrm{h}(y) \geq \mathrm{h}(x)$, and since $y \lhd x$ implies $\mathrm{h}(y) \leq \mathrm{h}(x)$, then x, y have the same height. We deduce that $x_1 \prec y$. If y has a second lower cover y' which is distinct from x_1, then y', x_1, x_2 is a clique of concurrent events. Hence, if such y' exists, then $y' = x_2$. Second, we remark that if $y, z \in C(x)$ and $x \in E_2$ then $y \mathrel{\overline{\frown}} z$: if $y' < y$ then $y' \leq x$ so that $x \mathrel{\overline{\frown}} z$ implies $y' \mathrel{\overset{\frown}{\smile}} z$, and symmetrically. It follows that for $x \in E_2$, $C(x)$ may have at most 2 elements. In particular, the restriction of $\lhd$ to E_2 is a 2-elimination ordering. $\square$ *Claim*

For $x \in E_1$ let $\rho(x) = \max\{\, z \in E \mid z \leq x, z \notin E_1 \,\}$ and $[x] = \{\, y \in E_1 \mid \rho(y) = \rho(x) \,\}$. Let $\mathcal{P}$ be the partition $\{\, E_0 \,\} \cup \{\, [x] \mid x \in E_1 \,\} \cup \{\, E_2 \,\}$. Since each $[x]$, $x \in E_1$, is a tree, the partition $\mathcal{P}$ is such that each equivalence class induces a 3-colorable subgraph of $\mathcal{G}(\mathcal{E})$.

Claim. The graph $\mathcal{G}(\mathcal{P}, \mathcal{E})$ is 4-colorable.

Since E_0 is isolated in $\mathcal{G}(\mathcal{P}, \mathcal{E})$, is it enough to prove that the subgraph of $\mathcal{G}(\mathcal{P}, \mathcal{E})$ induced by the trees $\{\, [x] \mid x \in E_1 \,\}$ is 3-colorable. Transport the linear ordering $\lhd$ to a linear ordering on the set of trees: $[y] \lhd [x]$ if and only if $\rho(y) \lhd \rho(x)$. Define $Q_{\lhd}([x])$ as usual, we claim that $Q_{\lhd}([x])$ may contain at most two trees.

We define a function $f : Q_{\lhd}([x]) \longrightarrow C(\rho(x))$ as follows. If $[y] \mathrel{\overline{\frown}} [x]$ and $[y] \lhd [x]$ then we can pick $y' \in [y]$ and $x' \in [x]$ such that $y' \mathrel{\overline{\frown}} x'$. We notice also that $y' \mathrel{\overline{\frown}} \rho(x)$: from $\rho(x) \leq x' \mathrel{\overline{\frown}} y'$, we deduce $\rho(x) \mathrel{\overline{\frown}} y'$ or $\rho(x) \leq y'$. The latter, however, implies $\rho(x) \leq \rho(y)$, by the definition of ρ, and this relation contradicts $\rho(y) \lhd \rho(x)$. Thus we let

$$f([y]) = \min\{\, z \mid \rho(y) \leq z \leq y' \text{ and } z \not\leq \rho(x) \,\}\,.$$

By definition, $f([y]) \mathrel{\overline{\frown}} \rho(x)$ and every lower cover of $f([y])$ is a lower cover of x. This clearly holds if $f([y]) \neq \rho(y)$, and if $f([y]) = \rho(y)$ then it holds since $\rho(y) \lhd \rho(x)$ implies $f([y]) \in Q_{\lhd}(x) \subseteq C(x)$ by the previous Claim. Thus the set $f(Q_{\lhd}(x))$ has cardinality at most 2 and, moreover, we claim that f is injective. Let us suppose that $f([y]) = f([z])$. If $f([y]) = \rho(y)$, then $f([z]) = \rho(z)$ as well and $[y] = [x]$. Otherwise $f([y]) = f([x])$ impliees $\rho(y) = \rho(f([y])) = \rho(f([z])) = \rho(z)$ and $[y] = [z]$. $\square$ *Claim*

Thus, by applying Proposition 4, we deduce that $\mathcal{G}(\mathcal{E})$ has a labelling with 12 letters. $\square$

Acknowledgement. We would like to thank Rémi Morin for introducing and guiding us to and through the realm of concurrency theory.

References

1. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part 1. Theor. Comput. Sci. 13, 85–108 (1981)
2. Winskel, G., Nielsen, M.: Models for concurrency. In: Handbook of logic in computer science, of Handb. Log. Comput. Sci., vol. 4, pp. 1–148. Oxford Univ. Press, New York (1995)

3. Winskel, G.: Event structure semantics for CCS and related languages. In: Nielsen, M., Schmidt, E.M. (eds.) Automata, Languages, and Programming. LNCS, vol. 140, pp. 561–576. Springer, Heidelberg (1982)
4. Varacca, D., Yoshida, N.: Typed event structures and the *pi*-calculus: Extended abstract. Electr. Notes Theor. Comput. Sci. 158, 373–397 (2006)
5. Faggian, C., Maurel, F.: Ludics nets, a game model of concurrent interaction. In: LICS, pp. 376–385. IEEE Computer Society Press, Los Alamitos (2005)
6. Melliès, P.A.: Asynchronous games 2: The true concurrency of innocence. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 448–465. Springer, Heidelberg (2004)
7. Diekert, V., Rozenberg, G. (eds.): The book of traces. World Scientific Publishing Co. Inc., River Edge, NJ (1995)
8. Pratt, V.: Modeling concurrency with partial orders. Internat. J. Parallel Programming 15(1), 33–71 (1986)
9. Rozoy, B., Thiagarajan, P.S.: Event structures and trace monoids. Theoret. Comput. Sci. 91(2), 285–313 (1991)
10. Rozoy, B.: On distributed languages and models for concurrency. In: Rozenberg, G. (ed.) Advances in Petri Nets 1992. LNCS, vol. 609, pp. 267–291. Springer, Heidelberg (1992)
11. Assous, M.R., Bouchitté, V., Charretton, C., Rozoy, B.: Finite labelling problem in event structures. Theor. Comput. Sci. 123(1), 9–19 (1994)
12. Arnold, A.: An extension of the notions of traces and of asynchronous automata. ITA 25, 355–396 (1991)
13. Hoogers, P.W., Kleijn, H.C.M., Thiagarajan, P.S.: An event structure semantics for general Petri nets. Theoret. Comput. Sci. 153(1-2), 129–170 (1996)
14. Dilworth, R.P.: A decomposition theorem for partially ordered sets. Ann. of Math. 51(2), 161–166 (1950)
15. Mycielski, J.: Sur le coloriage des graphs. Colloq. Math. 3, 161–162 (1955)
16. Niebert, P., Qu, H.: The implementation of mazurkiewicz traces in poem. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 508–522. Springer, Heidelberg (2006)
17. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: Probst, D.K., von Bochmann, G. (eds.) CAV 1992. LNCS, vol. 663, pp. 164–177. Springer, Heidelberg (1993)
18. Niebert, P., Huhn, M., Zennou, S., Lugiez, D.: Local first search - a new paradigm for partial order reductions. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 396–410. Springer, Heidelberg (2001)
19. Grätzer, G.: The congruences of a finite lattice. Birkhäuser Boston Inc., Boston, MA (2006) A proof-by-picture approach.
20. Santocanale, L.: Topological properties of event structures. GETCO06 (August 2006)
21. Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pacific J. Math. 15, 835–855 (1965)
22. Zykov, A.A.: On some properties of linear complexes. Mat. Sbornik N.S. 24(66), 163–188 (1949)

Causal Message Sequence Charts*

Thomas Gazagnaire[1], Blaise Genest[2], Loïc Hélouët[3], P.S. Thiagarajan[4], and Shaofa Yang[3]

[1] IRISA/ENS Cachan, Campus de Beaulieu, 35042 Rennes Cedex, France
thomas.gazagnaire@irisa.fr
[2] IRISA/CNRS, Campus de Beaulieu, 35042 Rennes Cedex, France
blaise.genest@irisa.fr
[3] IRISA/INRIA, Campus de Beaulieu, 35042 Rennes Cedex, France
{loic.helouet, shaofa.yang}@irisa.fr
[4] School of Computing, NUS, Singapore
thiagu@comp.nus.edu.sg

Abstract. Scenario languages based on Message Sequence Charts (MSCs) and related notations have been widely studied in the last decade [14,13,2,9,6,12,8]. The high expressive power of scenarios renders many basic problems concerning these languages undecidable. The most expressive class for which several problems are known to be decidable is one which possesses a behavioral property called "existentially bounded". However, scenarios outside this class are frequently exhibited by asynchronous distributed systems such as sliding window protocols. We propose here an extension of MSCs called Causal Message Sequence Charts, which preserves decidability without requiring existential bounds. Interestingly, it can also model scenarios from sliding window protocols. We establish the expressive power and complexity of decision procedures for various subclasses of Causal Message Sequence Charts.

1 Introduction

Scenario languages based on Message Sequence Charts (MSCs) have met considerable interest in the last ten years. The attractiveness of this notation can be explained by two major characteristics. Firstly, from the engineering point of view, MSCs have a simple and appealing graphical representation based on just a few concepts: processes, messages and internal actions. Secondly, from a mathematical standpoint, scenario languages admit an elegant formalization: they can be defined as languages generated by finite state automata over an alphabet of MSCs. These automata are usually called High-level Message Sequence Charts (HMSCs) [10].

An MSC is a restricted kind of labelled partial order and an HMSC is a generator of a (usually infinite) set of MSCs, that is, a language of MSCs. For example, the MSC M shown in Figure 2 is a member of the MSC language generated by the HMSC of Figure 1 while the MSC N shown in Figure 2 is not.

* Work supported by the INRIA-NUS Associated Team CASDS and the ANR projects DOTS.

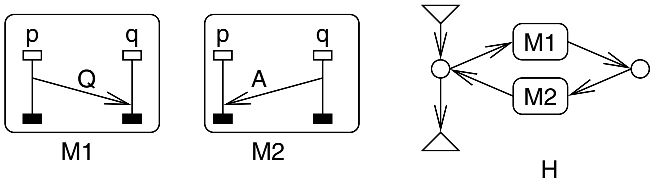


Fig. 1. An HMSC over two MSCs

HMSCs are very expressive and hence a number of basic problems associated with them cannot be solved effectively. For instance, it is undecidable whether two HMSCs generate the same collection of MSCs [14], or whether an HMSC generates a regular MSC language (an MSC language is regular if the collection of all the linearizations of all the MSCs in the language is a regular string language in the usual sense). Consequently, subclasses of HMSCs have been identified [13,2,6] and studied.

On the other hand, a basic limitation of HMSCs is that their MSC languages are finitely generated. More precisely, each MSC in the language can be defined as the sequential composition of elements chosen from a fixed finite set of MSCs [12]. However, the behaviours of many protocols constitute MSC languages that are *not* finitely generated. This occurs for example with scenarios generated by the alternating bit protocol. Such protocols can induce a collection of braids like N in Figure 2 which cannot be finitely generated.

One way to handle this is to work with so called safe (realizable) *Compositional* HMSCs (CHMCs, for short) in which message emissions and receptions are decoupled in individual MSCs but matched up at the time of composition, so as to yield a (complete) MSC. CHMSCs are however notationally awkward and do not possess the visual appeal of HMSCs. Furthermore, several positive results on HMSCs rely on a decomposition of MSCs into atoms (the minimal non-trivial MSCs) [9,12,6], which does not apply for CHMSCs, and results in a higher complexity [5]. It is also worth noting that without the restriction to safety (realizability), compositional HMSC languages embed the full expressive power of communicating automata [3] and consequently inherit all their undecidability results.

This paper proposes another approach to extend HMSCs in a tractable manner. The key feature is to allow the events belonging to a lifeline to be partially ordered. More specifically, we extend the notion of an MSC to that of *causal* MSC in which the events belonging to each lifeline (process), instead of being linearly ordered, are allowed to be partially ordered. To gain modelling power, we do not impose any serious restrictions on the nature of this partial order. Secondly, we assume a suitable Mazurkiewicz trace alphabet [4] for each lifeline and use this to define a composition operation for causal MSCs. This leads to the notion of causal HMSCs which generate tractable languages of causal MSCs.

A causal HMSC is *a priori* not existentially bounded in the sense defined in [5]. Informally, this property of an MSC language means that there is a uniform upper bound K such that for every MSC in the language *there exists* an execution

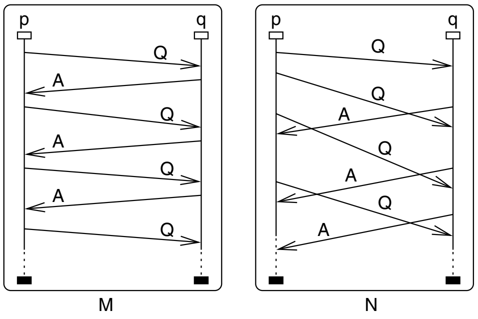


Fig. 2. Two MSCs M and N

along which—from start to finish—all FIFO channels remain K-bounded. Since this property fails, in general, for causal MSC languages, the main method used to gain decidability for safe CMSCs [5] is not applicable. Instead, to characterize regularity and decidability of certain subclasses of causal HMSCs, we need to generalize the method of [13] and of [6] in a non-trivial way.

In the next section we introduce causal MSCs and causal HMSCs. We also define the means for associating an ordinary MSC language with a causal HMSC. In the subsequent section we develop the basic theory of causal HMSCs. In section 4, we identify the property called "window-bounded", an important ingredient of the "braid"-like MSC languages generated by many protocols. Basically, this property bounds the number of messages a process p can send to a process q without waiting for an acknowledgement to be received. We then show that one can decide if a given causal HMSC generates a window-bounded MSC language. In section 5 we compare the expressive power of languages based on causal HMSCs with other known HMSC-based language classes. Proofs are omitted due to lack of space, but can be found in the full version of the paper available at: `www.irisa.fr/distribcom/Personal_Pages/helouet/Papers/full_concur07.pdf` .

2 MSCs, Causal MSCs and Causal HMSCs

Through the rest of the paper, we fix a finite nonempty set $\mathcal{P}$ of process names with $|\mathcal{P}| > 1$. For convenience, we let p, q range over $\mathcal{P}$ and drop the subscript $p \in \mathcal{P}$ when there is no confusion. We also fix finite nonempty sets *Msg*, *Act* of message types and internal action names respectively. We define the alphabets $\Sigma_! = \{p!q(m) \mid p, q \in \mathcal{P}, p \neq q, m \in Msg\}$, $\Sigma_? = \{p?q(m) \mid p, q \in \mathcal{P}, p \neq q, m \in Msg\}$, and $\Sigma_{act} = \{p(a) \mid p \in \mathcal{P}, a \in Act\}$. The letter $p!q(m)$ means the sending of message m from p to q; $p?q(m)$ the reception of message m at p from q; and $p(a)$ the execution of internal action a by process p. Let $\Sigma = \Sigma_! \cup \Sigma_? \cup \Sigma_{act}$. We define the *location* of a letter α in Σ, denoted $loc(\alpha)$, by $loc(p!q(m)) = p = loc(p?q(m)) = loc(p(a))$. For each process p in $\mathcal{P}$, we set $\Sigma_p = \{\alpha \in \Sigma \mid loc(\alpha) = p\}$. In order to define a concatenation operation for causal MSCs, we fix a family of Mazurkiewicz trace alphabets $\{(\Sigma_p, I_p)\}_{p \in \mathcal{P}}$ ([4]), one for each p. That is, $I_p \subseteq \Sigma_p \times \Sigma_p$ is a reflexive and symmetric relation, called the independence relation. We denote the dependence relation $(\Sigma_p \times \Sigma_p) - I_p$ by D_p. Following the usual definitions of Mazurkiewicz traces, for each (Σ_p, I_p), the associated trace equivalence relation $\sim_p$ over $\Sigma_p^\star$ is the least equivalence relation such that, for any u, v in $\Sigma_p^\star$ and α, β in Σ_p, $\alpha\ I_p\ \beta$ implies $u\alpha\beta v \sim_p u\beta\alpha v$. Equivalence classes of $\sim_p$ are called *traces*. For u in $\Sigma_p^\star$, we let $[u]_p$ denote the trace containing u.

Definition 1. *A* causal MSC *over* $(\mathcal{P}, \Sigma)$ *is a structure* $B = (E, \lambda, \{\sqsubseteq_p\}_{p\in\mathcal{P}}, \ll)$, *where* E *is a finite nonempty set of* events, $\lambda : E \to \Sigma$ *is a labelling function. And the following conditions hold:*

- *For each process* p, $\sqsubseteq_p \subseteq E_p \times E_p$ *is a* partial order, *where* $E_p = \{e \in E \mid \lambda(e) \in \Sigma_p\}$. *We let* $\widehat{\sqsubseteq}_p \subseteq E_p \times E_p$ *denote the least relation such that* $\sqsubseteq_p$ *is the reflexive and transitive closure of* $\widehat{\sqsubseteq}_p$.
- $\ll \subseteq E_! \times E_?$ *is a bijection, where* $E_! = \{e \in E \mid \lambda(e) \in \Sigma_!\}$ *and* $E_? = \{e \in E \mid \lambda(e) \in \Sigma_?\}$. *For each* $(e, e') \in \ll$, $\lambda(e) = p!q(m)$ *iff* $\lambda(e') = q?p(m)$.
- *The transitive closure of the relation* $\big(\bigcup_{p\in\mathcal{P}} \sqsubseteq_p\big) \cup \ll$, *denoted* $\leq$, *is a partial order.*

For each p, the relation $\sqsubseteq_p$ dictates the "causal" order in which events of E_p may be executed. The relation $\ll$ identifies pairs of message-emission and message-reception events. We say $\sqsubseteq_p$ *respects* the trace alphabet (Σ_p, I_p) iff for any $e, e' \in E_p$, the following hold: (i) $\lambda(e)\ D_p\ \lambda(e')$ implies $e \sqsubseteq_p e'$; (ii) $e\ \widehat{\sqsubseteq}_p\ e'$ implies $\lambda(e)\ D_p\ \lambda(e')$. The causal MSC B is said to respect $\{(\Sigma_p, I_p)\}$ iff $\sqsubseteq_p$ respects (Σ_p, I_p) for every p. In order to gain modelling power, we allow each $\sqsubseteq_p$ to be *any* partial order, not necessarily respecting (Σ_p, I_p). We say that the causal MSC B is *FIFO*[1] iff for any $(e, f) \in \ll$, $(e', f') \in \ll$ such that $\lambda(e) = \lambda(e') = p!q(m)$ (and thus $\lambda(f) = \lambda(f') = q?p(m)$), we have either $e \sqsubseteq_p e'$ and $f \sqsubseteq_q f'$; or $e' \sqsubseteq_p e$ and $f' \sqsubseteq_q e'$. Note that we do not demand *a priori* that a causal MSC must be FIFO.

Let $B = (E, \lambda, \{\sqsubseteq_p\}, \ll)$ be a causal MSC. A *linearization* of B is a word $a_1 a_2 \ldots a_\ell$ over Σ such that $E = \{e_1, \ldots, e_\ell\}$ with $\lambda(e_i) = a_i$ for each i; and $e_i \leq e_j$ implies $i \leq j$ for any i, j. We let $Lin(B)$ denote the set of linearizations of B. Clearly, $Lin(B)$ is nonempty. We set $Alph(B) = \{\lambda(e) \mid e \in E\}$, and $Alph_p(B) = Alph(B) \cap \Sigma_p$ for each p.

The leftmost part of Figure 3 depicts a causal MSC M. In this diagram, we enclose events of each process p in a vertical box and show the partial order $\sqsubseteq_p$ in the standard way. In case $\sqsubseteq_p$ is a total order, we place events of p along a vertical line with the minimum events at the top and omit the box. In particular, in M, the two events on p are not ordered (i.e. $\widehat{\sqsubseteq}_p$ is empty) and $\sqsubseteq_q$ is a total order. Members of $\ll$ are indicated by horizontal or downward-sloping arrows

[1] There are two notions of FIFOness for MSCs in the literature. One allows overtaking of messages with different message types via the same channel, while the other does not. As our results hold for both notions, we choose the more permissive one.

labelled with the transmitted message. Both words $p!q(Q).q!p(A).q?p(Q).p?q(A)$ and $q!p(A).p?q(A).p!q(Q).q?p(Q)$ are linearizations of M.

An *MSC* $B = (E, \lambda, \{\sqsubseteq_p\}_{p\in\mathcal{P}}, \lll)$ is defined in the same way as a causal MSC except that every $\sqsubseteq_p$ is required to be a *total order*. In an MSC B, the relation $\sqsubseteq_p$ must be interpreted as the visually observed order of events in one sequential execution of p. Let $B' = (E', \lambda', \{\sqsubseteq'_p\}, \lll')$ be a causal MSC. Then we say the MSC B is a *visual extension* of B' if $E' = E$, $\lambda' = \lambda$, $\sqsubseteq'_p \subseteq \sqsubseteq_p$ and $\lll' = \lll$. We let $Vis(B')$ denote the set of visual extensions of B'. In Figure 3, $Vis(M)$ consists of MSCs $M1, M2$.

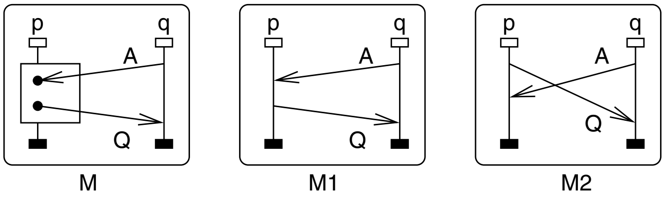


Fig. 3. A causal MSC M and its visual extensions $M1, M2$

We shall now define the concatenation operation of causal MSCs using the trace alphabets $\{(\Sigma_p, I_p)\}$.

Definition 2. *Let* $B = (E, \lambda, \{\sqsubseteq_p\}, \lll)$ *and* $B' = (E', \lambda', \{\sqsubseteq'_p\}, \lll')$ *be causal MSCs. We define the* concatenation *of* B *with* B', *denoted by* $B \circledcirc B'$, *as the causal MSC* $B'' = (E'', \lambda'', \{\sqsubseteq''_p\}, \lll'')$ *where*

- E'' *is the disjoint union of* E *and* E'. λ'' *is given by:* $\lambda''(e) = \lambda(e)$ *if* $e \in E$ *and* $\lambda''(e) = \lambda'(e)$ *if* $e \in E$. *And* $\lll'' = \lll \cup \lll'$.
- *For each* p, $\sqsubseteq''_p$ *is the transitive closure of*

$$\sqsubseteq_p \bigcup \sqsubseteq'_p \bigcup \{(e, e') \in E_p \times E'_p \mid \lambda(e)\ D_p\ \lambda'(e')\} \ .$$

Clearly $\circledcirc$ is a well-defined and associative operation. Note that in case B and B' are MSCs and $D_p = \Sigma_p \times \Sigma_p$ for every p, then the result of $B \circledcirc B'$ is the asynchronous concatenation (also called weak sequential composition) of B with B' [15], which we denote by $B \circ B'$. We also remark that the concatenation of causal MSCs is different from the concatenation of traces. The concatenation of trace $[u]_p$ with $[v]_p$ is the trace $[uv]_p$. However, a causal MSC B need not respect $\{(\Sigma_p, I_p)\}$. Consequently, for a process p, $Lin(B)$ may contain a word u such that the projection of u on $Alph_p(B)$ is *not* a trace.

We can now define causal HMSCs.

Definition 3. *A* causal HMSC *over* $(\mathcal{P}, \{(\Sigma_p, I_p)\})$ *is a structure* $H = (N, N_{in}, \mathcal{B}, \longrightarrow, N_{fi})$ *where* N *is a finite nonempty set of* nodes, $N_{in} \subseteq N$ *the set of initial nodes,* $\mathcal{B}$ *a finite nonempty set of causal MSCs,* $\longrightarrow \subseteq N \times \mathcal{B} \times N$ *the transition relation, and* $N_{fi} \subseteq N$ *the set of final nodes.*

A *path* in the causal HMSC H is a sequence $\rho = n_0 \xrightarrow{B_1} n_1 \xrightarrow{B_2} \cdots \xrightarrow{B_\ell} n_\ell$. If $n_0 = n_\ell$, then we say ρ is a cycle. The path ρ is *accepting* iff $n_0 \in N_{in}$ and $n_\ell \in N_{fi}$. The causal MSC generated by ρ, denoted $\circledcirc(\rho)$, is $B_1 \circledcirc B_2 \circledcirc \cdots \circledcirc B_\ell$. Note that the concatenation operation $\circledcirc$ is associative. We let $CaMSC(H)$ denote the set of causal MSCs generated by accepting paths of H. We also set $Vis(H) = \bigcup\{ Vis(M) \mid M \in CaMSC(H)\}$ and $Lin(H) = \bigcup\{Lin(M) \mid M \in CaMSC(H)\}$. Obviously, $Lin(H)$ is also equal to $\bigcup\{Lin(M) \mid M \in Vis(H)\}$. We shall refer to $CaMSC(H)$, $Vis(H)$, $Lin(H)$, respectively, as the causal language, visual language and linearization language of H.

An *HMSC* $H = (N, N_{in}, \mathcal{B}, \longrightarrow, N_{fi})$ is defined in the same way as a causal HMSC except that $\mathcal{B}$ is a finite set of MSCs and every MSC in $\mathcal{B}$ is FIFO. A path ρ of H generates an MSC by concatenating the MSCs along ρ. We let $Vis(H)$ denote the set of MSCs generated by accepting paths of H with $\circ$, and call $Vis(H)$ the visual language of H. Recall that an MSC language (i.e. a collection of MSCs) L is *finitely generated* [12] iff there exists a finite set X of MSCs satisfying the condition: for each MSC B in L, there exist $B_1, \ldots, B_\ell$ in X such that $B = B_1 \circ \cdots \circ B_\ell$. Many protocols exhibit scenario collections that are *not* finitely generated. For example, sliding window protocols can generate arbitarily large MSCs repeating the communication behaviour shown in MSC N of Figure 2. One basic limitation of HMSCs is that their visual languages are *finitely generated*. In contrast, the visual language of a causal HMSC is *not* necessarily finitely generated. For instance, suppose we view H in Figure 1 as a causal HMSC by considering $M1$,$M2$ as causal MSCs and associating H with the independence relations given by: $I_p = \{((p!q(Q), p?q(A)), (p?q(A), p!q(Q)))\}$ and $I_q = \emptyset$. Then clearly $Vis(H)$ is not finitely generated, as it contains infinitely many MSCs similar to N of Figure 2.

3 Regularity and Model-Checking for Causal HMSCs

3.1 Semantics for Causal HMSCs

As things stand, a causal HMSC H defines three syntactically different languages, namely its linearization language $Lin(H)$, its visual language (MSC) language $Vis(H)$ and its causal MSC language $CaMSC(H)$. The next proposition shows that they are also semantically different in general. It also identifies the restrictions under which they match semantically.

Proposition 1. *Let H, H' be causal HMSCs over the* same *family of trace alphabets $\{(\Sigma_p, I_p)\}$. Consider the following three hypotheses: (i) $CaMSC(H) = CaMSC(H')$; (ii) $Vis(H) = Vis(H')$; and (iii) $Lin(H) = Lin(H')$. Then we have:*

- *(i) $\implies$ (ii) and (ii) $\implies$ (iii); but the converses do not hold in general.*
- *If every causal MSC labelling transitions of H, H' respects $\{(\Sigma_p, I_p)\}$, then (ii) $\implies$ (i).*
- *If every causal MSC labelling transitions of H, H' is FIFO, then (iii) $\implies$ (ii).*

For most purposes, the relevant semantics for a causal HMSC seems to be its visual language.

3.2 Regular Sets of Linearizations

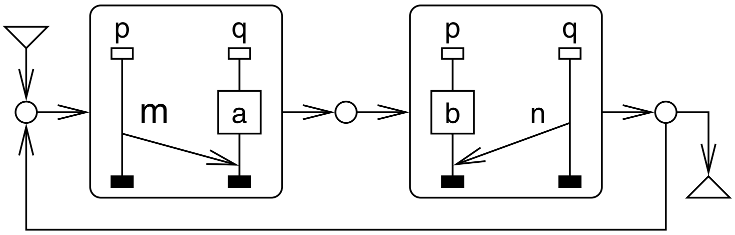
It is undecidable in general whether an HMSC has a regular linearization language [13]. In the literature, a subclass of HMSCs called regular [13] (or bounded [2]) HMSCs, has been identified. The linearization language of every regular HMSC is regular. And one can effectively whether an HMSC is in the subclass of regular HMSCs. We extend these results to causal HMSCs. First, let us recall the notions of connectedness from Mazurkiewicz traces theory [4], and of communication graphs [2,13,6]. Let $p \in \mathcal{P}$, and $B = (E, \lambda, \{\sqsubseteq_p\}, \ll)$ be a causal MSC. We say that $\Gamma \subseteq \Sigma_p$ is D_p-connected iff the (undirected) graph $(\Gamma, D_p \cap (\Gamma \times \Gamma))$ is connected. Moreover, we define the *communication graph* of B, denoted by CG_B, to be the directed graph $(Q, \rightsquigarrow)$, where $Q = \{p \in \mathcal{P} \mid E_p \neq \emptyset\}$ and $\rightsquigarrow \subseteq Q \times Q$ is given by $(p,q) \in \rightsquigarrow$ iff $\ll \cap (E_p \times E_q) \neq \emptyset$. Now we say the causal MSC B is *tight* iff its communication graph CG_B is connected and for every p, $Alph_p(B)$ is D_p-connected. We say the causal MSC B is *rigid* iff (i) B is FIFO; (ii) CG_B is strongly connected; and (iii) for every p, $Alph_p(B)$ is D_p-connected. We will focus here on rigidity and study the notion of tightness in section 3.3.

Let $H = (N, N_{in}, \mathcal{B}, \longrightarrow, N_{fi})$ be a causal HMSC. We say that H is *regular* iff for every cycle ρ in H, the causal MSC $\circledcirc(\rho)$ is rigid. For instance, the simple protocol modeled by the causal HMSC of Figure 4, is regular, since the only cycle is labeled by two local events a, b, one message from p to q and one message from q to p. The communication graph associated to this cycle is then strongly connected, $p!q(m) - b - p?q(n)$ on process p is connected, and $q!p(n) - a - q?p(m)$ on process q is connected. Equivalently, H is regular iff for every strongly connected subgraph G of H with $\{B_1, \ldots, B_\ell\}$ being the set of causal MSCs appearing in G, we have $B_1 \circledcirc \ldots \circledcirc B_\ell$ is rigid. Note that the rigidity of $B_1 \circledcirc \ldots \circledcirc B_\ell$ does not depend on the order in which $B_1, \ldots, B_\ell$ are listed. This leads to a co-NP-complete algorithm to test whether a causal HMSC is regular.

In the same way, we say that H is *globally-cooperative* iff for every strongly connected subgraph G of H with $\{B_1, \ldots, B_\ell\}$ being the set of causal MSCs appearing in G, we have that $B_1 \circledcirc \ldots \circledcirc B_\ell$ is tight.

Theorem 1. *Let $H = (N, N_{in}, \mathcal{B}, \longrightarrow, N_{fi})$ be a regular causal HMSC. Then $Lin(H)$ is a regular subset of $\Sigma^\star$, that is, we can build a finite state automaton $\mathcal{A}_H$ over Σ that recognizes $Lin(H)$. Furthermore, $\mathcal{A}_H$ has at most $\left(|N|^2 \cdot 2^{|\Sigma|} \cdot 2^{|N|\cdot|\Sigma|\cdot 2^m}\right)^{|N|\cdot|\Sigma|\cdot 2^m}$ states, where $m = \max\{|B| \mid B \in \mathcal{B}\}$.*

Intuitively, for a word σ in $\Sigma^\star$, $\mathcal{A}_H$ guesses an accepting path ρ of H and checks whether σ is in $Lin(\circledcirc(\rho))$. Upon reading a prefix σ' of σ, $\mathcal{A}_H$ memorizes a sequence of subpaths of ρ from which σ' was "linearized". The crucial step is to ensure that at any time, it suffices to remember a *bounded* number of such subpaths, and moreover, a *bounded* amount of information for each subpath.



$I_p = \{ (p?q(n), p!q(m)), (p!q(m), p?q(n)) \}$

$I_q = \{ (q?p(m), q!p(n)), (q!p(n), q?p(m)) \}$

Fig. 4. A regular causal HMSC which is not finitely generated

3.3 Inclusion and Intersection Non-emptiness of Causal HMSCs

As the linearization languages of regular causal HMSCs are regular, verification for regular causal HMSCs can be effectively solved. It is natural to ask whether we can still obtain positive results of verification beyond the subclass of regular causal HMSCs. As for HMSCs, one can show that for a suitable choice of K, the set of K-bounded linearizations of any globally cooperative HMSC is regular, and this is sufficient for effective verification [5]. Unfortunately, this result uses Kuske's encoding [11] into traces that is based on the existence of an (existential) bound on communication. Consequently, this technique does not apply to globablly cooperative causal HMSCs, as the visual language of a causal HMSC needs not be existentially bounded. For instance, consider the causal HMSC H of Figure 5. It is globally cooperative (but not regular), and its visual language contains MSCs shown in the right part of Figure 5: in order to receive the first message from p to r, the message from p to q and the message from q to r have to be sent and received. Hence every message from p to r has to be sent before receiving the first message from p to r, which means that H is not existentially bounded.

It is known that problems of inclusion, intersection non-emptiness and equality of visual languages of HMSCs are undecidable [13]. Clearly, these undecidability results also apply to causal HMSCs. In [13], decidability results for inclusion and intersection non-emptiness of globally cooperative HMSCs are established. Our goal here is to extend these results to globally cooperative causal HMSCs.

We shall adapt the notion of atoms [1,9] and the techniques from [6]. Let us first introduce a notion of decomposition of causal MSCs into basic parts.

Definition 4. *A causal MSC B is a* basic part *(w.r.t. the trace alphabets $\{(\Sigma_p, I_p)\}$) if there do not exist causal MSCs B_1, B_2 such that $B = B_1 \circledcirc B_2$.*

Note that we require that the set of events of a causal MSC is not empty. Now for a causal MSC B, we define a *decomposition* of B to be a sequence $B_1 \cdots B_\ell$ of basic parts such that $B = B_1 \circledcirc \cdots \circledcirc B_\ell$. For a set $\mathcal{B}$ of basic parts, we associate a trace alphabet $(\mathcal{B}, I_\mathcal{B})$ (w.r.t. the trace alphabets $\{(\Sigma_p, I_p)\}$) where $I_\mathcal{B}$ is given by:

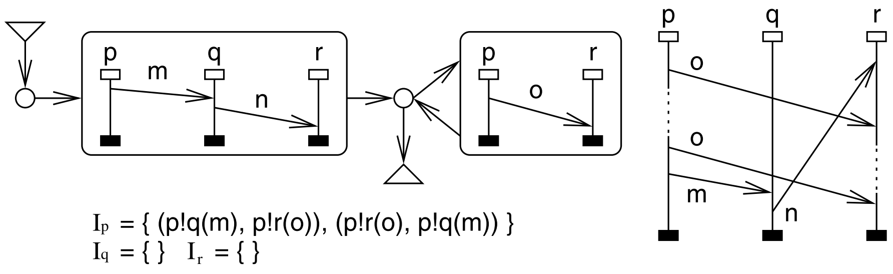


Fig. 5. A globally-cooperative causal HMSC that is not existentially bounded

$B\ I_{\mathcal{B}}\ B'$ iff for every p, for every $\alpha \in Alph_p(B)$, for every $\alpha' \in Alph_p(B')$, it is the case that $\alpha\ I_p\ \alpha'$. We let $\sim_{\mathcal{B}}$ be the corresponding trace equivalence relation and denote the trace containing a sequence $u = B_1 \ldots . B_\ell$ in $\mathcal{B}^\star$ by $[u]_{\mathcal{B}}$ (or simply $[u]$). For a language $L \subseteq \mathcal{B}^\star$, we define its trace closure $[L]_{\mathcal{B}} = \bigcup\{[u]_{\mathcal{B}} \mid u \in L\}$.

Proposition 2. *For a given causal MSC B, we can effectively construct the smallest finite set of basic parts, denoted $Basic(B)$, such that every decomposition of B is in $Basic(B)^\star$. Further, the set of decompositions of B forms a trace of $(Basic(B), I_{Basic(B)})$.*

We briefly describe the algorithm for constructing $Basic(B)$, which is analogous to technique in [9]. Let $B = (E, \lambda, \{\sqsubseteq_p\}, \ll)$. We consider the undirected graph (E, R), where R is the symmetric closure of $\ll \cup \big(\bigcup_{p \in \mathcal{P}} R'_p \cup R''_p\big)$, where $R'_p = \{(e, e') \in E_p \times E_p \mid e \sqsubseteq_p e' \text{ and } \lambda(e)\ I_p\ \lambda(e')\}$ and $R''_p = \{(e, e') \in E_p \times E_p \mid e \not\sqsubseteq_p e' \text{ and } e' \not\sqsubseteq_p e \text{ and } \lambda(e)\ D_p\ \lambda(e')\}$. Each basic part in $Basic(B)$ can be formed from a connected component of (E, R) and thus $Basic(B)$ can be constructed in quadratic time.

In view of Proposition 2, we assume through the rest of this section that every transition of a causal HMSC H is labelled by a basic part. Clearly this incurs no loss of generality, since we can simply decompose each causal MSC in H into basic parts and decompose any transition of H into a sequence of transitions labeled by these basic parts. Given a causal HMSC H, we let $Basic(H)$ be the set of basic parts labelling transitions of H. Proposition 2 implies that a causal MSC is uniquely defined by its basic part decomposition. Then instead of the linearization language we can use the *basic part language* of H, denoted by $BP(H) = \{B_1 \ldots B_\ell \in Basic(H)^\star \mid B_1 \circledcirc \ldots \circledcirc B_\ell \in CaMSC(H)\}$. Notice that $BP(H) = [BP(H)]$ by Proposition 2, that is, $BP(H)$ is closed by commutation. We can also view H as a finite state automaton over the alphabet $Basic(H)$, and denote by $\mathcal{L}_{Basic}(H) = \{B_1 \cdots B_\ell \in Basic(H)^\star \mid n_0 \xrightarrow{B_1} n_1 \cdots \xrightarrow{B_\ell} n_\ell$ is an accepting path of H.$\}$ its associated (regular) language. We now relate $BP(H)$ and $\mathcal{L}_{Basic}(H)$.

Proposition 3. *Let H be a causal HMSC. Then $BP(H) = [\mathcal{L}_{Basic}(H)]$.*

Assuming we know how to compute the trace closure of the regular language $\mathcal{L}_{Basic}(H)$, we can obtain $BP(H)$ with the help of Proposition 3. In general, we cannot effectively compute this language. However if H is globally cooperative, then $[\mathcal{L}_{Basic}(H)]$ is regular and a finite state automaton recognizing $[\mathcal{L}_{Basic}(H)]$ can be effectively constructed [4,13]. Considering globally cooperative causal HMSCs as finite state automata over basic parts, we can apply [13] to obtain the following decidability and complexity results:

Theorem 2. *Let H, H' be causal HMSCs over the* same *family of trace alphabets $\{(\Sigma_p, I_p)\}$. Suppose H' is globally cooperative. Then we can build a finite state automaton $\mathcal{A}'$ over Basic(H') such that $\mathcal{L}_{Basic}(A') = [\mathcal{L}_{Basic}(H')]$. And $\mathcal{A}'$ has at most $2^{O(n \cdot b)}$ states, where n is the number of nodes in H and b is the number of basic parts in Basic(H). Consequently, the following problems are decidable:*

(i) Is CaMSC$(H) \subseteq$ CaMSC(H')?
(ii) Is CaMSC$(H) \cap$ CaMSC$(H') = \emptyset$?

Furthermore, the complexity of (i) is PSPACE-complete and that of (ii) is EXPSPACE-complete.

The above theorem shows that we can model check a causal HMSC against a globally cooperative causal HMSC specification. Note that we can only apply Theorem 2 to two causal HMSCs over the *same* family of trace alphabets. If the causal HMSCs H, H' in theorem 2 satisfy the additional condition that every causal MSCs labeling the transitions of H and H' respects $\{(\Sigma_p, I_p)\}$, then we can compare the visual languages *Vis*(H) and *Vis*(H'), thanks to Proposition 1. On the other hand, when two causal HMSCs are defined with different families of trace alphabets, the only possible comparison between them seems to be on their linearization languages. Consequently, we would need to work with regular causal HMSCs.

4 Window-Bounded Causal HMSCs

One of the chief attractions of causal MSCs is they enable the specification of behaviors containing braids of arbitrary size such as those generated by sliding windows protocols. Very often, sliding windows protocols appear in a situation where two processes p and q exchange bidirectional data. Messages from p to q are of course used to transfer information, but also to acknowledge messages from q to p. If we abstract the type of messages exchanged, these protocols can be seen as a series of query messages from p to q and answer messages from q to p. Implementing a sliding window means that a process may send several queries in advance without needing to wait for an answer to each query before sending the next query. Very often, these mechanisms tolerate losses, i.e. the information sent is stored locally, and can be retransmitted if needed (as in the alternating bit protocol). To avoid memory leaks, the number of messages that can be sent in advance is often bounded by some integer k, that is called the size of the sliding window. Note however that for scenario languages defined using causal HMSCs, such window sizes do not always exist. This is the case for example for the causal HMSC depicted in Figure 1 with independence relations $I_p = \{((p!q(Q), p?q(A)), (p?q(A), p!q(Q)))\}$

and $I_q = \{((q?p(Q), q!p(A)), (q!p(A), q?p(Q))\}$. The language generated by this causal HMSC contains scenarios where an arbitrary number of messages from p to q can cross an arbitrary number of messages from q to p. A question that naturally arises is to know if the number of messages crossings is bounded by some constant in all the executions of a protocol specified by a causal HMSC. In what follows, we define these crossings, and show that their boundedness is a decidable problem.

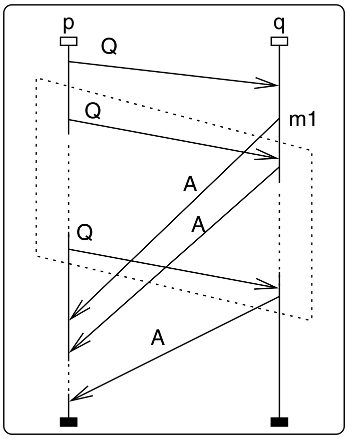


Fig. 6. Window of message m_1

Definition 5. *Let $M = (E, \lambda, \{\sqsubseteq_p\}, \ll)$ be an MSC For a message (e, f) in M, that is, $(e, f) \in \ll$, we define the* window *of (e, f), denoted $W_M(e, f)$, as the set of messages $\{(e', f') \in \ll \mid loc(\lambda(e')) = loc(\lambda(f))$ and $loc(\lambda(f')) = loc(\lambda(e))$ and $e \leq f'$ and $e' \leq f\}$.*

We say that a causal HMSC H is *K-window-bounded* iff for every $M \in Vis(H)$ and for every message (e, f) of M, it is the case that $|W_M(e, f)| \leq K$. H is said to be window-bounded iff H is K-window-bounded for some K.

Figure 6 illustrates notion of window, where the window of the message m_1 is symbolized by the area delimited by dotted lines. It consists of all but the first message Q from p to q. Clearly, the causal HMSC H of Figure 1 is not window-bounded. We now describe an algorithm to effectively check whether a causal HMSC is window bounded. It builds a finite state automaton whose states remember the labels of events that must appear in the *future* of messages (respectively in the *past*) in any MSC of $Vis(H)$.

Formally, for a causal MSC $B = (E, \lambda, \{\sqsubseteq_p\}, \ll)$ and $(e, f) \in \ll$ a message of B, we define the future and past of (e, f) in B as follows:

$$Future_B(e, f) = \{a \in \Sigma \mid \exists x \in E, f \leq x \wedge \lambda(x) = a\}$$
$$Past_B(e, f) = \{a \in \Sigma \mid \exists x \in E, x \leq e \wedge \lambda(x) = a\}$$

In Figure 6, $Past_B(m_1) = \{p!q(Q), q?p(Q), q!p(A)\}$.

Proposition 4. *Let $B = (E, \lambda, \{\sqsubseteq_p\}, \ll)$ and $B' = (E', \lambda', \{\sqsubseteq'_p\}, \ll')$ be two causal MSCs, and let $m \in \ll$ be a message of B. Then we have:*

$$\begin{aligned} \mathit{Future}_{B \circledcirc B'}(m) = \mathit{Future}_B(m) \cup \{a' \in \Sigma \mid \exists x, y \in E' \\ \exists a \in \mathit{Future}_B(m) \text{ s.t. } \lambda(y) = a' \wedge x \leq' y \wedge a \; D_{loc(a)} \; \lambda(x)\} \end{aligned}$$

Let $H = (N, N_{in}, \mathcal{B}, \longrightarrow, N_{fi})$ be a causal HMSC. Consider a path ρ of H with $\circledcirc(\rho) = B_1 \circledcirc \cdots \circledcirc B_\ell$ and a message m in B_1. Then Proposition 4 implies the sequence of sets $\mathit{Future}_{B_1}(m)$, $\mathit{Future}_{B_1 \circledcirc B_2}(m)$, ..., $\mathit{Future}_{B_1 \circledcirc \cdots \circledcirc B_\ell}(m)$ is non-decreasing. Furthermore, these sets can be computed on the fly, that is with a finite state automaton. Similar arguments hold for the past sets. Now consider a message (e, f) in a causal MSC B labelling some transition t of H. With the above observation on *Future* and *Past*, we can show that, if there is a bound $K_{(e,f)}$ such that the window of a message (e, f) in the causal MSC generated by any path containing t is bounded by $K_{(e,f)}$, then $K_{(e,f)}$ is at most $b|N|(|\Sigma| + 1)$ where $b = \max\{|B| \mid B \in \mathcal{B}\}$. Further, we can effectively determine whether such a bound $K_{(e,f)}$ exists by constructing a finite state automaton whose states memorize the future and past of (e, f). Thus we have the following:

Theorem 3. *Let $H = (N, N_{in}, \mathcal{B}, \longrightarrow, N_{fi})$ be a causal HMSC. If H is window-bounded, then H is K-window-bounded, where K is at most $b|N|(|\Sigma| + 1)$ with $b = \max\{|B| \mid B \in \mathcal{B}\}$. Further, we can effectively determine whether H is window-bounded in time $O(s \cdot |N|^2 \cdot 2^{|\Sigma|})$, where s is the sum of the sizes of causal MSCs in $\mathcal{B}$.*

5 Relationship with Other Scenario Models

We compare here the expressive power of other HMSC-based scenario languages with causal HMSCs in terms of their visual languages. We consider first HMSCs. Two important strict HMSC subclasses are (i) *regular* [13] (also called bounded in [2]) HMSCs which ensure that the linearizations form a regular set and (ii) *globally-cooperative* HMSCs [6], which ensure that for a suitable choice of K, the set of K-bounded linearizations form a regular set. By definition, causal HMSCs, regular causal HMSCs and globally-cooperative causal HMSCs extend respectively HMSCs, regular HMSCs and globally-cooperative HMSCs.

Figure 5 shows a globally-cooperative causal HMSC which is not in the subclass of regular causal HMSCs. Thus, regular causal HMSCs form a strict subclass of globally-cooperative causal HMSCs. Trivially, globally-cooperative causal HMSCs are a strict subclass of causal HMSCs. Figure 4 displays a regular causal HMSC whose visual language is not finitely generated. It follows that (regular/globally-cooperative) causal HMSCs are strictly more powerful than (regular/globally-cooperative) HMSCs.

Another extension of HMSCs is *Compositional* HMSCs [7], or CHMSCs for short. CHMSCs generalize HMSCs by allow dangling message-sending and message-reception events, i.e. where the message pairing relation $\ll$ is only a partial non-surjective mapping contained in $E_! \times E_?$. The concatenation of two Compositional MSCs $M \circ M'$ performs the instance-wise concatenation as for MSCs, and computes a new message pairing relation $\ll''$ defined over $(E_! \cup E'_!) \times (E_? \cup E'_?)$ extending $\ll \cup \ll'$, and preserving the FIFO ordering of messages of the same content (actually, in the definition of [7], there is no channel content).

A CHMSC H generates a set of MSCs, denoted $\mathit{Vis}(H)$ by abuse of notation, obtained by concatenation of MSCs along a path of the graph. With this definition, some path of a CHMSC may not generate any correct MSC. Moreover, a path of a CHMSC generates at most one MSC. The class of CHMSC for which each path generates exactly one MSC is called *safe* CHMSC, still a strict extension over HMSCs. Regular and globally cooperative HMSCs have also their strict extensions in terms of safe CHMSCs, namely as regular CHMSC and globally cooperative CHMSCs.

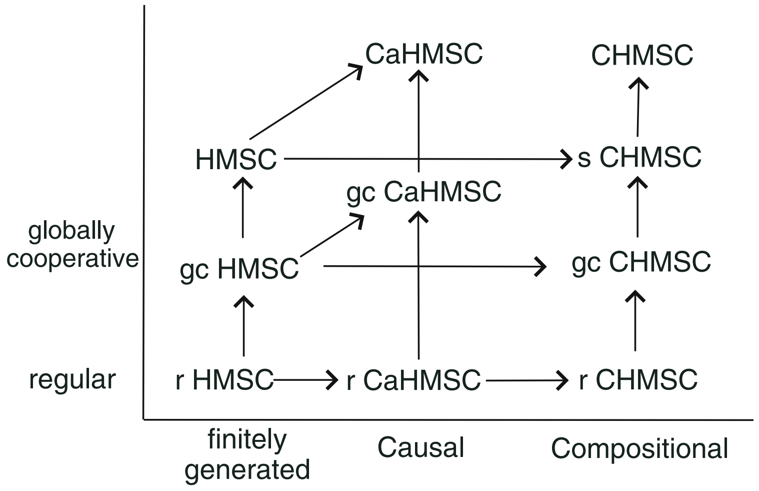


Fig. 7. Comparison of Scenario languages

It is not hard to build a regular Compositional HMSC H with $\mathit{Vis}(H) = \{M_i \mid i = 0, 1, \ldots\}$ where each M_i consists of an emission event e from p to r, then a sequence of i blocks of three messages: a message from p to q followed by a message from q to r then a message from r to p. And at last the reception event on r from p matching e. That is, H is not finitely generated. A causal HMSC cannot generate the same language. Assume for contradiction, a causal HMSC G with $\mathit{Vis}(G) = \mathit{Vis}(H)$. Let k be the number of messages of the biggest causal MSC which labels a transition of G. We know that M_{k+1} is in $\mathit{Vis}(G)$, hence $M_{k+1} \in \mathit{Vis}(\circledcirc(\rho))$ for some accepting path ρ of G. Let $N_1, \ldots, N_\ell$ be causal MSCs along ρ, where $\ell \geq 2$ because of the size k. It also means that there exist $N'_1 \in \mathit{Vis}(N_1)$, …, $N'_\ell \in \mathit{Vis}(N_\ell)$ such that $N'_1 \circ \cdots \circ N'_m \in \mathit{Vis}(G)$. Thus, $N_1 \circ \cdots \circ N_\ell = M_j$ for some j, a contradiction since M_j is a basic part (i.e. cannot be the concatenation of two MSCs). That is (regular) compositional HMSCs are not included into causal HMSCs. On the other hand, regular causal HMSCs have a regular set of linearizations (Theorem 1). Also by the results in [8], it is immediate that the class of visual languages of regular compositional HMSCs captures all the MSC languages that have a regular set of linearizations. Hence the class of regular causal HMSCs is included into the class of regular compositional HMSCs. Last, we already know with Figure 5 that globally-cooperative causal HMSCs are not necessarily existentially bounded, hence they are not included into safe Compositional HMSC. Furthermore, globally-cooperative causal HMSCs are not included into CHMSCs because the former can generate MSCs that are *not* FIFO.

The relationships among these scenario models are summarized by Figure 7, where arrows denote *strict* inclusion of visual languages. Two classes are incomparable if they are not connected by a transitive sequence of arrows. We use the abbreviation *r* for regular, *gc* for globally-cooperative, *s* for safe, *CaHMSC* for causal HMSCs and *CHMSC* for compositional HMSCs.

6 Conclusion

We have defined an extension of HMSC called causal HMSC that allows the definition of braids, such as those appearing in sliding window protocols. We also identified in this setting, many subclasses of scenarios that were defined for HMSCs which have decidable verification problems. An interesting class that emerges is globally-cooperative causal HMSCs. This class is incomparable with safe Compositional HMSCs because the former can generate scenario collections that are not existentially bounded. Yet, decidability results of model checking can be obtained for this class.

An interesting open problem is deciding whether the visual language of a causal HMSC is finitely generated. Yet another interesting issue is to consider the class of causal HMSCs whose visual languages are window-bounded. The set of behaviours generated by these causal HMSCs seems to exhibit a kind of regularity that could be exploited. Finally, designing suitable machine models (along the lines of Communicating Finite Automata [3]) is also an important future line of research.

References

1. Ahuja, M., Kshemkalyani, A.D., Carlson, T.: A basic unit of computation in distributed sytems. In: Proc. of ICDS'90, pp. 12–19 (1990)
2. Alur, R., Yannakakis, M.: Model checking of message sequence charts. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 114–129. Springer, Heidelberg (1999)
3. Brand, D., Zafiropoulo, P.: On communicating finite state machines. Technical Report RZ1053, IBM Zurich Research Lab (1981)
4. Diekert, V., Rozenberg, G. (eds.): The Book of Traces. World Scientific, Singapore (1995)
5. Genest, B., Kuske, D., Muscholl, A.: A Kleene theorem and model checking for a class of communicating automata. Information and Computation 204(6), 920–956 (2006)

6. Genest, B., Muscholl, A., Seidl, H., Zeitoun, M.: Infinite-state high-level MSCs: Model-checking and realizability. Journal of Computer and System Sciences 72(4), 617–647 (2006)
7. Gunter, E., Muscholl, A., Peled, D.: Compositional message sequence charts. In: Margaria, T., Yi, W. (eds.) ETAPS 2001 and TACAS 2001. LNCS, vol. 2031, Springer, Heidelberg (2001)
8. Henriksen, J.G., Mukund, M., Narayan Kumar, K., Sohoni, M., Thiagarajan, P.S.: A theory of regular MSC languages. Information and Computation 202(1), 1–38 (2005)
9. Hélouët, L., Le Maigat, P.: Decomposition of message sequence charts. In: Proc. of SAM'00 (2000)
10. ITU-TS: ITU-TS Recommendation Z.120: Message Sequence Chart (MSC). ITU-TS (1999)
11. Kuske, D.: Regular sets of infinite message sequence charts. Information and Computation 187(1), 80–109 (2003)
12. Morin, R.: Recognizable sets of message sequence charts. In: Alt, H., Ferreira, A. (eds.) STACS 2002. LNCS, vol. 2285, pp. 523–534. Springer, Heidelberg (2002)
13. Muscholl, A., Peled, D.: Message sequence graphs and decision problems on Mazurkiewicz traces. In: Kutyłowski, M., Wierzbicki, T., Pacholski, L. (eds.) MFCS 1999. LNCS, vol. 1672, Springer, Heidelberg (1999)
14. Muscholl, A., Peled, D., Su, Z.: Deciding properties for message sequence charts. In: Nivat, M. (ed.) ETAPS 1998 and FOSSACS 1998. LNCS, vol. 1378, pp. 226–242. Springer, Heidelberg (1998)
15. Reniers, M.: Message Sequence Chart: Syntax and Semantics. PhD thesis, Eindhoven University of Technology (1999)

Checking Coverage for Infinite Collections of Timed Scenarios*

S. Akshay[1,3], Madhavan Mukund[2], and K. Narayan Kumar[2]

[1] LSV, ENS Cachan, France
akshay@lsv.ens-cachan.fr
[2] Chennai Mathematical Institute, Chennai, India
{madhavan,kumar}@cmi.ac.in
[3] Institute of Mathematical Sciences, Chennai, India

Abstract. We consider message sequence charts enriched with timing constraints between pairs of events. As in the untimed setting, an infinite family of time-constrained message sequence charts (TC-MSCs) is generated using an HMSC—a finite-state automaton whose nodes are labelled by TC-MSCs. A timed MSC is an MSC in which each event is assigned an explicit time-stamp. A timed MSC *covers* a TC-MSC if it satisfies all the time constraints of the TC-MSC. A natural recognizer for timed MSCs is a message-passing automaton (MPA) augmented with clocks. The question we address is the following: given a timed system specified as a time-constrained HMSC H and an implementation in the form of a timed MPA $\mathcal{A}$, is every TC-MSC generated by H covered by some timed MSC recognized by $\mathcal{A}$? We give a complete solution for locally synchronized time-constrained HMSCs, whose underlying behaviour is always regular. We also describe a restricted solution for the general case.

1 Introduction

In a distributed system, several agents interact with each other to generate a global behaviour. The interaction between these agents is usually specified in terms of scenarios, using message sequence charts (MSCs) [7].

We consider scenarios extended with timing constraints, called time-constrained MSCs (TC-MSCs). In a TC-MSC, we associate lower and upper bounds on the time interval between certain pairs of events. TC-MSCs are a natural and useful extension of the untimed notation for scenarios, because protocol specifications typically include timing requirements for message exchanges, as well as descriptions of how to recover from timeouts.

As an implementation model for timed distributed systems, we use communicating finite-state machines equipped with clocks, called timed message-passing automata (timed MPAs). Clock constraints are used to guard transitions and specify location invariants, as in other models of timed automata [3]. Just as the

* Partially supported by *Timed-DISCOVERI*, a project under the Indo-French Networking Programme.

runs of timed automata can be described in terms of timed words, the interactions exhibited by timed MPAs can be described using timed MSCs—MSCs in which each event is assigned an explicit timestamp.

Scenario specifications are typically incomplete and can be classified into two categories, positive and negative. Positive scenarios are those that the system should execute while negative scenarios indicate undesirable behaviours. This leads to the *scenario matching* problem: given a distributed system and a set of positive (negative) scenarios, does the system exhibit (avoid) these scenarios? In the untimed setting, efficient algorithms for the scenario matching problem have been identified in [10]. An automated approach is proposed in [5].

The timed analogue of scenario matching is *coverage*. A timed MSC T *covers* a TC-MSC specification M if the timestamps on the events in T satisfy the constraints specified in M. In general, a TC-MSC is covered by infinitely many timed MSCs. The coverage problem is to check whether the set of timed MSCs generated by a timed MPA cover all the TC-MSCs in the specification.

For finite sets of TC-MSCs, this problem reduces to the intersection of timed regular languages. In this case, checking coverage can be automated using the modelchecker UPPAAL for timed systems, as described in [4].

In this paper, we consider the coverage problem for infinite collections of TC-MSCs. A standard way to generate an infinite set of MSCs is to use a High-level Message Sequence Chart (HMSC) [8]. In its most basic form, an HMSC is a finite directed graph, called a Message Sequence Graph (MSG), with each node labelled by an MSC. We label nodes in an MSGs with TC-MSCs and add time constraints across edges, resulting in a structure called a time-constrained MSG (TC-MSG). A TC-MSG defines a collection of TC-MSCs by concatenating the TC-MSCs labeling each path from an initial node to a terminal node.

Formally, coverage asks whether for every TC-MSC M generated by a TC-MSG, there is a timed MSC exhibited by the system that covers M. Since the set of TC-MSCs generated by a TC-MSG is infinite, it turns out that coverage can no longer be reduced to a simple intersection of timed regular languages.

We describe an algorithm to solve the coverage problem for *locally synchronized* TC-MSGs—those for which the underlying behaviour is guaranteed to be regular [6]. Our approach consists of "guiding" the timed MPA implementation to follow the TC-MSG specification in such a way that we can reduce the problem to *untimed* language inclusion. This allows us to solve the coverage problem both for positive and negative specifications. For arbitrary TC-MSGs, a fully general guided simulation can result in non-regular behaviours. However, we can adapt our approach to solve the coverage problem for TC-MSCs that are executed node by node in the underlying TC-MSG.

The paper is organized as follows. We begin with some preliminaries about MSCs and MSGs. In the next section, we describe how to attach timing information to scenario specifications. Section 4 formally describes timed message-passing automata. With this background, we formally describe the coverage problem in Section 5. Our solution is described in Section 6. We conclude with a brief discussion.

2 Preliminaries on MSCs

2.1 Message Sequence Charts

Let $\mathcal{P} = \{p, q, r, \ldots\}$ be a finite set of processes (agents) that communicate through messages via reliable FIFO channels using a finite set of message types $\mathcal{M}$. For $p \in \mathcal{P}$, let $Act_p = \{p!q(m), p?q(m) \mid p \neq q \in \mathcal{P}, m \in \mathcal{M}\}$ be the set of communication actions for p. The action $p!q(m)$ is read as p *sends the message* m *to* q and the action $p?q(m)$ is read as p *receives the message* m *from* q. We set $Act = \bigcup_{p \in \mathcal{P}} Act_p$. We also denote the set of *channels* by $Ch = \{(p,q) \mid p \neq q\}$.

Labelled posets. An Act-labelled poset is a structure $M = (E, \leq, \lambda)$ where $(E, \leq)$ is a poset and $\lambda : E \to Act$ is a labelling function.

For $e \in E$, let $\downarrow e = \{e' \mid e' \leq e\}$. For $X \subseteq E$, $\downarrow X = \cup_{e \in X} \downarrow e$. We call $X \subseteq E$ a *prefix* of M if $X = \downarrow X$. For $p \in \mathcal{P}$ and $a \in Act$, we set $E_p = \{e \mid \lambda(e) \in Act_p\}$ and $E_a = \{e \mid \lambda(e) = a\}$, respectively.

For each $(p, q) \in Ch$, we define a relation $<_{pq}$ as follows, to captures the fact that channels are FIFO with respect to each message—if $e <_{pq} e'$, the message m read by q at e' is the one sent by p at e.

$$e <_{pq} e' \overset{\triangle}{=} \lambda(e) = p!q(m),\ \lambda(e') = q?p(m) \text{ and } |\downarrow e \cap E_{p!q(m)}| = |\downarrow e' \cap E_{q?p(m)}|$$

Finally, for each $p \in \mathcal{P}$, we define the relation $\leq_{pp} = (E_p \times E_p) \cap \leq$, with $<_{pp}$ standing for the largest irreflexive subset of $\leq_{pp}$.

Definition 1. *An MSC (over $\mathcal{P}$) is a* finite *Act-labelled poset $M = (E, \leq, \lambda)$ that satisfies the following conditions.*

1. *Each relation $\leq_{pp}$ is a linear order.*
2. *If $p \neq q$ then for each $m \in \mathcal{M}$, $|E_{p!q(m)}| = |E_{q?p(m)}|$.*
3. *If $e <_{pq} e'$, then $|\downarrow e \cap \left(\bigcup_{m \in \mathcal{M}} E_{p!q(m)}\right)| = |\downarrow e' \cap \left(\bigcup_{m \in \mathcal{M}} E_{q?p(m)}\right)|$.*
4. *The partial order $\leq$ is the reflexive, transitive closure of the relation $\bigcup_{p,q \in \mathcal{P}} <_{pq}$.*

The second condition ensures that every message sent along a channel is received. The third condition says that every channel is FIFO across all messages.

In diagrams, the events of an MSC are presented in *visual order*. The events of each process are arranged in a vertical line and messages are displayed as horizontal or downward-sloping directed edges. Fig. 1 shows an example with three processes $\{p, q, r\}$ and six events $\{e_1, e_1', e_2, e_2', e_3, e_3'\}$ corresponding to three messages—m_1 from p to q, m_2 from q to r and m_3 from p to r.

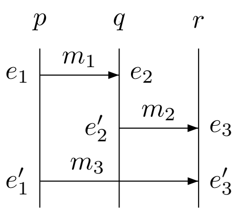


Fig. 1. An MSC

For an MSC $M = (E, \leq, \lambda)$, we let $\text{lin}(M) = \{\lambda(\pi) \mid \pi \text{ is a linearization of } (E, \leq)\}$. For instance, $p!q(m_1)\ q?p(m_1)\ q!r(m_2)\ p!r(m_3)\ r?q(m_2)\ r?p(m_3)$ is one linearization of the MSC in Fig. 1.

MSC languages. An *MSC language* is a set of MSCs.
We can also regard an MSC language $\mathcal{L}$ as a word language L over Act consisting of all linearizations of the MSCs in $\mathcal{L}$. For an MSC language $\mathcal{L}$, we set $\text{lin}(\mathcal{L}) = \bigcup\{\text{lin}(M) \mid M \in \mathcal{L}\}$.

Definition 2. *An MSC language $\mathcal{L}$ is said to be a* regular MSC language *if the word language* $\text{lin}(\mathcal{L})$ *is a regular language over* Act.

Let M be an MSC and $B \in \mathbb{N}$. We say that $w \in \text{lin}(M)$ is B-bounded if for every prefix v of w and for every channel $(p,q) \in Ch$, $\sum_{m\in\mathcal{M}} |\pi_{p!q(m)}(v)| - \sum_{m\in\mathcal{M}} |\pi_{q?p(m)}(v)| \leq B$, where $\pi_\Gamma(v)$ denotes the projection of v on $\Gamma \subseteq Act$. This means that along the execution of M described by w, no channel ever contains more than B-messages. We say that M is (universally) B-bounded if every $w \in \text{lin}(M)$ is B-bounded. An MSC language $\mathcal{L}$ is B-bounded if every $M \in \mathcal{L}$ is B-bounded. Finally, $\mathcal{L}$ is bounded if it is B-bounded for some B.

We then have the following result [6].

Theorem 1. *If an MSC language $\mathcal{L}$ is regular then it is bounded.*

2.2 Message Sequence Graphs

Message sequence graphs (MSGs) are finite directed graphs with designated initial and terminal vertices. Each vertex in an MSG is labelled by an MSC. The edges represent (asynchronous) MSC concatenation, in which one MSC is "pasted" below the other. Formally, MSC concatenation is defined as follows.

Let $M_1 = (E^1, \leq^1, \lambda_1)$ and $M_2 = (E^2, \leq^2, \lambda_2)$ be a pair of MSCs such that E^1 and E^2 are disjoint. The *(asynchronous) concatenation* of M_1 and M_2 yields the MSC $M_1 \circ M_2 = (E, \leq, \lambda)$ where $E = E^1 \cup E^2$, $\lambda(e) = \lambda_i(e)$ if $e \in E^i$, $i \in \{1,2\}$, and $\leq\, = (\leq^1 \cup \leq^2 \cup \bigcup_{p\in\mathcal{P}} E^1_p \times E^2_p)^*$.

A *Message Sequence Graph* is a structure $G = (Q, \rightarrow, Q_{in}, Q_F, \Phi)$, where Q is a finite and nonempty set of states, $\rightarrow\, \subseteq Q \times Q$, $Q_{in} \subseteq Q$ is a set of initial states, $Q_F \subseteq Q$ is a set of final states and Φ labels each state with an MSC.

A *path* π through an MSG G is a sequence $q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_n$ such that $(q_{i-1}, q_i) \in\, \rightarrow$ for $i \in \{1,2,\ldots,n\}$. The MSC generated by π is $M(\pi) = M_0 \circ M_1 \circ M_2 \circ \cdots \circ M_n$, where $M_i = \Phi(q_i)$. A path $\pi = q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_n$ is a *run* if $q_0 \in Q_{in}$ and $q_n \in Q_F$. The language of MSCs accepted by G is $L(G) = \{M(\pi) \mid \pi \text{ is a run through } G\}$. We say that an MSC language $\mathcal{L}$ is *MSG-definable* if there exists and MSG G such that $\mathcal{L} = L(G)$.

An example of an MSG is depicted in Fig. 2. The initial state is marked $\Rightarrow$ and the final state has a double circle. The language $\mathcal{L}$ defined by this MSG is *not* regular: $\mathcal{L}$ projected to $\{p!q(m), r!s(m)\}^*$ consists of $\sigma \in \{p!q(m), r!s(m)\}^*$ such that $|\pi_{p!q(m)}(\sigma)| = |\pi_{r!s(m)}(\sigma)| \geq 1$, which is not a regular string language.

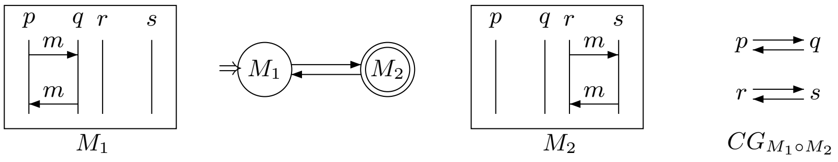


Fig. 2. A message sequence graph

In general, it is undecidable whether an MSG describes a regular MSC language [6]. However, a sufficient condition for the MSC language of an MSG to be regular is that the MSG be *locally synchronized*.

Communication graph. For an MSC $M = (E, \leq, \lambda)$, let CG_M, *the communication graph of* M, be the directed graph $(\mathcal{P}, \mapsto)$ where:

- $\mathcal{P}$ is the set of processes of the system.
- $(p, q) \in \mapsto$ iff there exists an $e \in E$ with $\lambda(e) = p!q(m)$.

M is said to be *com-connected* if CG_M consists of one nontrivial strongly connected component and isolated vertices.

Locally synchronized MSGs. The MSG G is *locally synchronized* [9] (or *bounded* [2]) if for every loop $\pi = q \to q_1 \to \cdots \to q_n \to q$, the MSC $M(\pi)$ is com-connected. In Fig. 2, $CG_{M_1 \circ M_2}$ is not com-connected, so the MSG is not locally synchronized. We have the following result for MSGs [2].

Theorem 2. *If G is locally synchronized, $L(G)$ is a regular MSC language.*

One of the factors contributing to the non-regularity of MSG-definable languages is that there is, in general, no bound on the asynchrony between processes. For instance, in Fig. 2, if we traverse the loop k times, we can identify a prefix of the MSC $\underbrace{M_1 \circ M_2 \circ \cdots \circ M_1 \circ M_2}_{k \text{ copies}}$ in which r and s are currently in the final copy of M_2 while p and q are in the first copy of M_1, at a distance $2k$. In locally synchronized MSGs, the gap between the first and last processes is always bounded. To formalize this, we define the active suffix of a path.

Active suffix. Let G be an MSG and $\pi = q_0 q_1 \ldots q_k$ a path through G. Let X be a prefix of $M(q_0) \circ M(q_1) \circ \cdots \circ M(q_k)$. For each process p, let $i_p \in \{0, 1, \ldots, k\}$ be the node such that the maximum p-event in X lies in $M(q_{i_p})$. By convention, if X has no p-events, $i_p = 0$. Let $i_{\min} = \min_{p \in \mathcal{P}} i_p$. The *active suffix* of π is defined to be the path $q_{i_{\min}} q_{i_{\min}+1} \ldots q_k$.

The following two facts about locally synchronized MSGs will be useful. The first follows from Theorems 1 and 2. The second fact is the key to the proof of Theorem 2 (see [6], Appendix A).

Corollary 1. *Let G be a locally synchronized MSG. Then we can effectively compute bounds $B, K \in \mathbb{N}$ such that:*

- *Every MSC in $L(G)$ is B-bounded.*
- *For every MSC $M \in L(G)$, for every prefix X of M, the length of the active suffix of X is bounded by K.*

3 Adding Time to Scenarios

3.1 Time-Constrained MSCs

A time-constrained MSC is an MSC annotated with time intervals between some pairs of events. For instance, consider the interaction between a user, an ATM

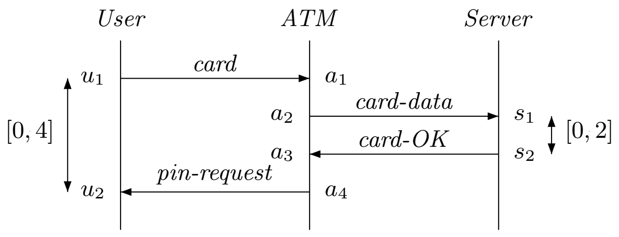


Fig. 3. A TC-MSC describing interaction with an ATM

and a server depicted in Fig. 3. This MSC has eight events generated by four messages, with time constraints between the event pairs (u_1, u_2) and (s_1, s_2). The constraint $[0, 2]$ on (s_1, s_2) specifies that the server is expected to respond to a request to authenticate an ATM card within 2 time units. Similarly, the constraint $[0, 4]$ on (u_1, u_2) specifies that a user will be asked to enter his PIN within 4 time units of inserting the card.

For simplicity, we assume that time intervals are bounded by natural numbers. For $m, n \in \mathbb{N}$, $[m, n]$ is the closed interval $\{x \in \mathbb{R}_{\geq 0} \mid m \leq x \leq n\}$, while (m, n) is the open interval $\{x \in \mathbb{R}_{\geq 0} \mid m < x < n\}$. As usual, we permit half-open intervals of the form $[m, n)$ and $(m, n]$. To specify an interval without an upper bound we write $[m, \infty)$ or (m, ∞). Let $\mathcal{I}$ denote the set of intervals.

Definition 3. *Let $M = (E, \leq, \lambda)$ be an MSC. An* interval constraint *is a tuple $\langle (e_1, e_2), I \rangle$ where $e_1, e_2 \in E$ with $e_1 \leq_{pp} e_2$ for some $p \in \mathcal{P}$ or $e_1 <_{pq} e_2$ for some channel $(p, q) \in Ch$ and $I \in \mathcal{I}$.*

The restrictions on e_1 and e_2 ensure that an interval constraint is either local to a process or describes a bound on the delivery time of a single message.

Definition 4. *A* time-constrained MSC (TC-MSC) *is a pair $\mathcal{T} = (M, \mathcal{EC})$ where $M = (E, \leq, \lambda)$ is an MSC and $\mathcal{EC} \subseteq (E \times E) \times \mathcal{I}$ is a set of interval constraints such that each pair (e_1, e_2) is mapped to at most one interval.*

3.2 Timed MSCs

In a timed MSC, events are explicitly time-stamped so that the ordering on the time-stamps respects the partial order on the events.

Definition 5. *A* timed MSC *is pair (M, τ) where $M = (E, \leq, \lambda)$ is an MSC and $\tau : E \longrightarrow \mathbb{R}_{\geq 0}$ assigns a nonnegative time-stamp to each event, such that for all $e_1, e_2 \in E$, if $e_1 \leq e_2$ then $\tau(e_1) \leq \tau(e_2)$.*

A timed MSC *covers* a TC-MSC if the time-stamps assigned to events respect the interval constraints specified in the TC-MSC. Let $r \in \mathbb{R}_{\geq 0}$ and $I \in \mathcal{I}$. We write $r \models I$ to denote that r lies in the interval specified by I.

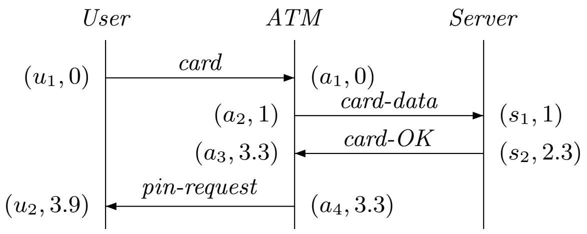


Fig. 4. A timed MSC describing interaction with an ATM

Definition 6. *Let $M = (E, \leq, \lambda)$ be an MSC, $\mathcal{T} = (M, \mathcal{EC})$ a TC-MSC and $M_\tau = (M, \tau)$ a timed MSC. M_τ is said to* cover *$\mathcal{T}$ if for each $\langle (e_1, e_2), I \rangle \in \mathcal{EC}, \tau(e_2) - \tau(e_1) \models I$.*

Fig. 4 shows a timed MSC that covers the TC-MSC in Fig. 3.

Let $M_\tau = (M, \tau)$ be a timed MSC, where $M = (E, \leq, \lambda)$. A *timed linearization* of M_τ is a sequence $(e_0, \tau(e_0))(e_1, \tau(e_1)) \cdots (e_n, \tau(e_n))$ where $e_0 e_1 \ldots e_m$ is a linearization of $(E, \leq)$ and $\tau(e_0) \leq \tau(e_1) \leq \cdots \leq \tau(e_n)$. As is the case with untimed MSCs, under the FIFO assumption for channels, a timed MSC can be faithfully reconstructed from any one of its timed linearizations.

3.3 Time-Constrained MSGs

A natural way to describe infinite families of TC-MSCs is to label the nodes of an MSG with TC-MSCs instead of normal MSCs. In addition, we permit local (process-wise) timing constraints along the edges of the MSG. A constraint for process p along an edge $q \to q'$ specifies a constraint between the final p-event of $M(q)$ and the initial p-event of $M(q')$, provided p actively participates in both these nodes. If p does not participate in either of these nodes, the constraint is ignored. We can think of each node in a TC-MSG as describing one phase of a communication protocol, with timing constraints along the edges specifying how long each process can wait between phases.

Definition 7. *A* time-constrained MSG (TC-MSG) *is a structure $G = (Q, \to, Q_{in}, Q_F, \Phi, EdgeC)$, where*

- *Q is a finite non-empty set of states with sets of initial and final states Q_{in} and Q_F, respectively, and $\to \subseteq Q \times Q$ is a transition relation, as in an MSG.*
- *Φ labels each node with a TC-MSC.*
- *$EdgeC \subseteq Q \times Q \times \mathcal{P} \times \mathcal{I}$ describes local constraints on the edges, with the restriction that $(q, q', p, I) \in EdgeC$ only if $q \to q'$ and each triple (q, q', p) is mapped to at most one interval.*

The definitions of *paths* and *runs* are the same for TC-MSGs as for MSGs. If $\pi = q_0 q_1 \ldots q_n$ is a path through G, the TC-MSC generated by π is denoted $M(\pi)$. To define $M(\pi)$, we begin with the TC-MSC $M_0 \circ M_1 \circ \cdots \circ M_n$, where $M_i = \Phi(q_i)$ for

$i \in \{0, 1, \dots, n\}$. For each edge $q_i \rightarrow q_{i+1}$, $0 \leq i < n$, if $(q_i, q_{i+1}, p, I) \in EdgeC$ we add a constraint I between the last p-event in M_i and the first p-event in M_{i+1}, provided p participates in both M_i and M_{i+1}.

The language $L(G)$ of TC-MSCs accepted by G is defined to be the set of TC-MSCs generated by the runs of G.

We define com-connectedness for TC-MSCs based on the communication graph just as we do for untimed MSCs. From this, we can define locally synchronized TC-MSGs in the same way as we do for MSGs.

4 Timed Message-Passing Automata

Message-passing automata are a natural machine model for recognizing MSCs. We extend the definition used in [6] to include clocks.

Definition 8. *Let $\mathcal{C}$ denote a finite-set of real-valued variables called* clocks. *A* clock constraint *is a conjunctive formula of the form $x \sim n$ or $x - y \sim n$ for $x, y \in \mathcal{C}$, $n \in \mathbb{N}$ and $\sim \in \{\leq, <, =, >, \geq\}$. Let $Form(\mathcal{C})$ denote the set of clock constraints over the set of clocks $\mathcal{C}$.*

Clock constraints will be used as guards in timed message-passing automata.

Definition 9. *A* clock assignment *for a set of clocks $\mathcal{C}$ is a function $v : \mathcal{C} \rightarrow \mathbb{R}_{\geq 0}$ that assigns a nonnegative real value to each clock in $\mathcal{C}$.*

A clock assignment v satisfies *a clock constraint φ, denoted $v \models \varphi$, if φ evaluates to true when we replace each clock x in φ by the value $v(x)$.*

Let $v : \mathcal{C} \rightarrow \mathbb{R}_{\geq 0}$ be a clock assignment. For $d \in \mathbb{R}_{\geq 0}$, $v + d$ denotes the clock assignment that maps each $x \in \mathcal{C}$ to $v(x) + d$. For $X \subseteq \mathcal{C}$, $v[X \leftarrow 0]$ is the clock assignment that agrees with v for $x \in \mathcal{C} \setminus X$ and maps all clocks in X to 0.

Definition 10. *A* timed message-passing automaton (timed MPA) *over Act with a set of clocks $\mathcal{C}$ is a structure $\mathcal{A} = (\{\mathcal{A}_p\}_{p \in \mathcal{P}}, Act, \mathcal{C})$. Each component $\mathcal{A}_p$ is of the form $(S_p, S_{in}^p, \rightarrow_p)$, where:*

- *S_p is a finite set of p-local states.*
- *$S_{in}^p \subseteq S_p$, is a set of initial states for p.*
- *$\rightarrow_p \subseteq S_p \times Form(\mathcal{C}) \times Act_p \times 2^{\mathcal{C}} \times S_p$ is the p-local transition relation.*

The local transition relation $\rightarrow_p$ specifies how the process p sends and receives messages. The transition $(s, \varphi, p!q(m), X, s')$ says that in state s, p can send the message m to q and move to state s'. This transition is *guarded* by the clock constraint φ—the transition is enabled only when the current values of all the clocks satisfy φ. The set X specifies the clocks whose values are reset to 0 when this transition is taken. Similarly, the transition $(s, \varphi, p?q(m), X, s')$ signifies that at state s, p can receive the message m from q and move to state s' provided the current clock values satisfy φ. Once again, all clocks in X are reset to 0.

To simplify the presentation, we have not included location invariants in our model. Location invariants are clock constraints attached to states that constrain the duration for which the automaton can remain in each state. Our results extend smoothly to timed MPAs equipped with location invariants.

Like timed automata, timed MPA can perform two types of moves: moves where the automaton does not change state and time elapses, and moves where some local component p changes state instantaneously as permitted by $\rightarrow_p$.

A global state of $\mathcal{A}$ is an element of $\prod_{p \in \mathcal{P}} S_p$. For a global state $\overline{s}$, $\overline{s}_p$ denotes the pth component of $\overline{s}$. A *configuration* is a triple $(\overline{s}, \chi, v)$ where $\overline{s}$ is a global state, $\chi : \mathit{Ch} \rightarrow \mathcal{M}^*$ is the *channel state* describing the message queue in each channel c and $v : \mathcal{C} \rightarrow \mathbb{R}_{\geq 0}$ is a clock assignment. An *initial configuration* of $\mathcal{A}$ is of the form $(\overline{s}_{in}, \chi_\varepsilon, v_0)$ where $\overline{s}_{in} \in \prod_{p \in \mathcal{P}} S^p_{in}$, $\chi_\varepsilon(c)$ is the empty string ε for every channel c and $v_0(x) = 0$ for every $x \in \mathcal{C}$.

The set of reachable configurations of $\mathcal{A}$, $\mathit{Conf}_\mathcal{A}$, is defined inductively, together with a transition relation $\Longrightarrow \subseteq \mathit{Conf}_\mathcal{A} \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times \mathit{Conf}_\mathcal{A}$.

– Every initial configuration $(\overline{s}_{in}, \chi_\varepsilon, v_0)$ is in $\mathit{Conf}_\mathcal{A}$.
– If $(\overline{s}, \chi, v) \in \mathit{Conf}_\mathcal{A}$ and $d \in \mathbb{R}_{\geq 0}$, then there is a global move $(\overline{s}, \chi, v) \xRightarrow{d} (\overline{s}, \chi, v + d)$ and $(\overline{s}, \chi, v + d) \in \mathit{Conf}_\mathcal{A}$.
– If $(\overline{s}, \chi, v) \in \mathit{Conf}_\mathcal{A}$ and $(\overline{s}_p, \varphi, p!q(m), X, \overline{s}'_p) \in \rightarrow_p$ such that v satisfies φ, there is a global move $(\overline{s}, \chi, v) \xRightarrow{p!q(m)} (\overline{s}', \chi', v[X \leftarrow 0])$ with $(\overline{s}', \chi', v[X \leftarrow 0]) \in \mathit{Conf}_\mathcal{A}$, where, for $r \neq p$, $\overline{s}_r = \overline{s}'_r$, $\chi'((p, q)) = \chi((p, q)) \cdot m$, and for $c \neq (p, q)$, $\chi'(c) = \chi(c)$.
– Similarly, if $(\overline{s}, \chi, v) \in \mathit{Conf}_\mathcal{A}$ and $(\overline{s}_p, \varphi, p?q(m), X, \overline{s}'_p) \in \rightarrow_p$ such that v satisfies φ, there is a global move $(\overline{s}, \chi, v) \xRightarrow{p?q(m)} (\overline{s}', \chi', v[X \leftarrow 0])$ with $(\overline{s}', \chi', v[X \leftarrow 0]) \in \mathit{Conf}_\mathcal{A}$, where, for $r \neq p$, $\overline{s}_r = \overline{s}'_r$, $\chi((q, p)) = m \cdot \chi'((q, p))$, and for $c \neq (q, p)$, $\chi'(c) = \chi(c)$.

Let $\mathrm{prf}(\sigma)$ denote the set of prefixes of a timed word $\sigma = (a_1, t_1)(a_2, t_2) \ldots (a_k, t_k) \in (\mathit{Act} \times \mathbb{R}_{\geq 0})^*$. A run of $\mathcal{A}$ over σ is a map $\rho : \mathrm{prf}(\sigma) \rightarrow \mathit{Conf}_\mathcal{A}$ such that $\rho(\varepsilon)$ is assigned an initial configuration $(\overline{s}_{in}, \chi_\varepsilon, v_0)$ and for each $\sigma' \cdot (a_i, t_i) \in \mathrm{prf}(\sigma)$, $\rho(\sigma') \xRightarrow{d_i}\xRightarrow{a_i} \rho(\sigma' \cdot (a_i, t_i))$ with $t_i = t_{i-1} + d_i$ and t_0 implicitly set to 0.

The run ρ is *complete* if $\rho(\sigma) = (s, \chi_\varepsilon, v)$ is a configuration in which all channels are empty. When a run on σ is complete, σ is a timed linearization of a timed MSC. We define $L(\mathcal{A}) = \{\sigma \mid \mathcal{A}$ has a complete run over $\sigma\}$. Thus, $L(\mathcal{A})$ corresponds to the set of timed linearizations of a collection of timed MSCs. Let $\mathcal{L}(\mathcal{A})$ be the language of timed MSCs corresponding to $L(\mathcal{A})$.

Fig. 5 shows a timed MPA along with two of the timed MSCs that it recognizes. In the timed MSCs, we have only written the time-stamps associated with the events and not the event names themselves. In this timed MPA, r sends a message m_1 to s. Process s replies with m_2 exactly 1 time unit after it receives m_1. If m_2 is received by r within 2 time units of its sending m_1, it sends m_3 and quits. Otherwise, if at least 2.2 time units go by before r receives m_2, it resends m_1. Note that there is no transition enabled in r for the interval $2 < x \leq 2.2$.

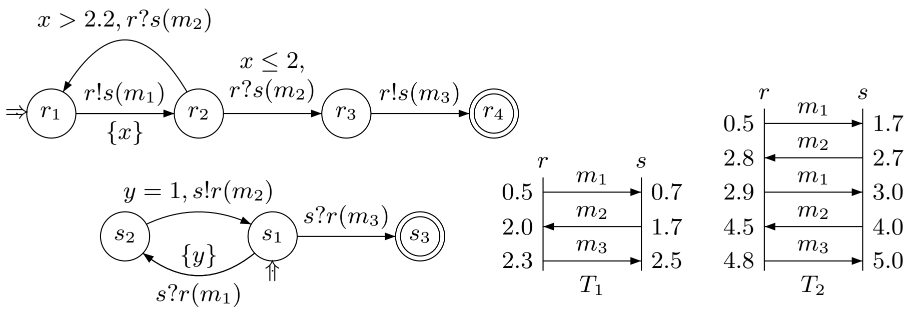


Fig. 5. A timed MPA and some timed MSCs that it recognizes

5 The Coverage Problem

We are interested in the following verification problem for timed scenario specifications. Let G be a TC-MSG and $\mathcal{A}$ a timed MPA. The *coverage problem* for G and $\mathcal{A}$ is to determine whether for each TC-MSC $M \in L(G)$, there is a timed MSC $T \in \mathcal{L}(\mathcal{A})$ such that T covers M. This is a natural verification problem when we interpret TC-MSGs as incomplete positive specifications.

For instance, consider the TC-MSG G in Fig. 6. In G, r sends a message m_1 to s that could be delayed by upto 3 time units, to which s replies after exactly 1 time unit. If the response m_2 from s reaches r within 2 time units from the time r sent m_1, r sends a final message m_3 and quits. Otherwise, r resends m_1. M_1 and M_2 (Fig. 6) are TC-MSCs in $L(G)$, generated by paths q_1q_2 and $q_1q_3q_1q_2$, respectively. These are covered by the timed MSCs T_1 and T_2 (Fig. 5) in $\mathcal{L}(\mathcal{A})$.

In the untimed case, scenario matching asks whether $L(G) \subseteq \mathcal{L}(\mathcal{A})$, where G is an MSG and $\mathcal{A}$ is an MPA. In the timed case, we cannot reduce coverage to language inclusion of timed MSCs. A TC-MSC M represents an infinite family of timed MSCs, each of which covers M. However, the implementation need not, in general, permit all these realizations. For instance, the timed MPA in Fig. 5 will not exhibit any timed MSC covering M_2 from Fig. 6 where the time difference between the first two p-events is 2.1, such as in the timed MSC T_2' in Fig. 6.

Another plausible approach is to treat this as a timed game between Spoiler, who picks a path in the TC-MSG G, and Duplicator, who picks a timed MSC in $\mathcal{L}(\mathcal{A})$ that covers the TC-MSC generated along the path chosen by Spoiler. At each step, Spoiler adds a node to the path in G. Duplicator has to match this move by extending the current timed MSC so that it stays in $\mathcal{L}(\mathcal{A})$ and covers the TC-MSC described by the extended path. However, a winning strategy in this game would have the following property: if two paths π_1 and π_2 have a common prefix π, then the timed MSC generated by Duplicator for the prefix π must be the same for the plays in which Spoiler generates π_1 and π_2. Notice that the paths that generate M_1 and M_2 in Fig. 6 share a common prefix, namely q_1. In any timed MSC that covers M_1, message m_1 must be delivered within 1 time unit whereas in any timed MSC that covers M_2, m_1 can only reach after 1 time unit. Hence, any timed MSC that covers $M(q_1)$ and can be extended to

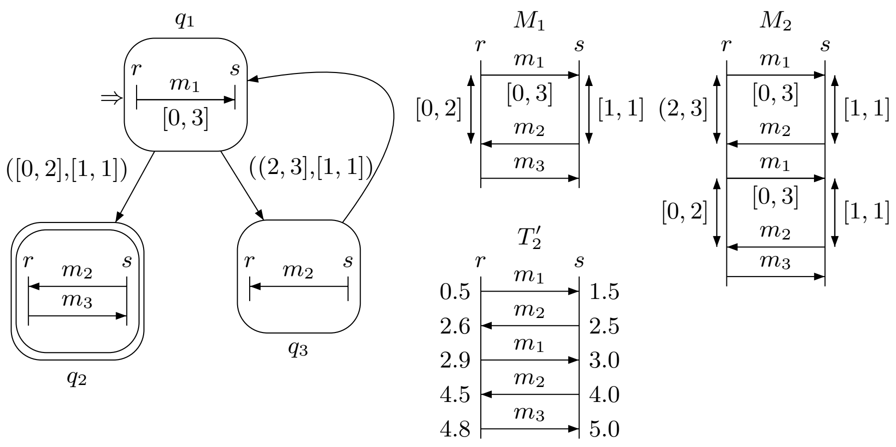


Fig. 6. The coverage problem

cover M_1 cannot simultaneously be extended to cover M_2. In other words, the game-theoretic formulation introduces too strict a correlation between the timed MSCs covering different paths through the TC-MSG.

These observations suggest that traditional approaches for scenario matching in the untimed case do not generalize to the coverage problem in the timed case. In the next section, we formulate a solution to the coverage problem for locally synchronized TC-MSGs.

6 Coverage for Locally Synchronized TC-MSGs

Theorem 3. *Let G be a locally synchronized TC-MSG and $\mathcal{A}$ a timed MPA. The coverage problem for G and $\mathcal{A}$ is decidable.*

Proof. Our proof strategy is as follows. From $\mathcal{A}$, we construct a timed automaton $\mathcal{B}$ that simulates the global behaviour of $\mathcal{A}$, restricted to runs that are consistent with paths through $L(G)$. In addition to the communication actions in Act, the timed automaton $\mathcal{B}$ also has moves labelled by nodes from G, indicating the path that it is following. As usual, we can use the region construction [1] to obtain an untimed regular language $Untime(L(\mathcal{B})) \subseteq (Act \cup Q)^*$. It will then turn out that $Untime(L(\mathcal{B}))$ projected onto Q precisely describes the set of paths through G that are covered by some timed MSC in $\mathcal{L}(\mathcal{A})$.

To check coverage, we just need to verify that the *node language* of G, $L_Q(G) = \{q_0 q_1 \dots q_n \in Q^* \mid q_0 \to q_1 \to \cdots \to q_n \text{ is a run}\}$, is included in $L_Q(\mathcal{B}) = \pi_Q(Untime(L(\mathcal{B})))$. This would imply that for every path in π through G, the TC-MSC $M(\pi)$ is covered by some timed MSC in $L(\mathcal{A})$.

There is, however, a complication. Some paths in G may define TC-MSCs that cannot be covered, because of self-contradictory timing constraints. These paths need not be covered by timed MSCs from $L(\mathcal{A})$. We cannot, therefore, directly compare $L_Q(G)$ with $L_Q(\mathcal{B})$. The solution is straightforward: we start with the trivial automaton $\mathcal{A}_U$ that recognizes Act^*, which can be regarded as a degenerate timed automaton with no timing constraints. To $\mathcal{A}_U$, we apply the same construction as we have done for $\mathcal{A}$. The resulting timed automaton $\mathcal{B}_U$ will mark out all paths π through G for which $M(\pi)$ can be covered by some timed MSC. We can now check whether $L_Q(\mathcal{B}_U) = \pi_Q(\mathit{Untime}(L(\mathcal{B}_U)))$ is included in $L_Q(\mathcal{B})$. Since both $L_Q(\mathcal{B}_U)$ and $L_Q(\mathcal{B})$ are regular languages, the result follows.

Constructing $\mathcal{B}$. Recall that $\mathcal{A} = (\{\mathcal{A}_p\}_{p\in\mathcal{P}}, Act, \mathcal{C})$, where each component $\mathcal{A}_p$ is of the form $(S_p, S_{in}^p, \rightarrow_p)$, as described in Section 4. The structure of the TC-MSG is given by $G = (Q, \rightarrow, Q_{in}, Q_F, \Phi, \mathit{EdgeC})$, as described in Section 3. Without loss of generality, we assume that all the events that occur in the TC-MSCs labelling nodes of G have distinct names, so that when we refer to a constraint $\langle (e, e'), I\rangle$, there is no ambiguity.

Alphabet. The alphabet of $\mathcal{B}$ is $Act \cup Q$, where Q is the set of nodes in G.

States. A state of $\mathcal{B}$ consists of the following components:

– $\overline{s} \in \prod_{p\in\mathcal{P}} S_p$, a global state of $\mathcal{A}$.
– $\chi : Ch \rightarrow \mathcal{M}^*$, the state of the channels.
– $\eta : Act \rightarrow \{0, \ldots, B-1\}$, a function to count occurrences of each action in Act modulo B.
– $\pi \in Q^*$, (the active suffix of) a path in G.
– $\rho : E_{M(\pi)} \rightarrow \{0, 1, 2\}$, a labelling function for the events in the TC-MSC $M(\pi)$. (Events labelled 0 are yet to be executed. Events labelled 1 represent the "active" frontier of events that will be executed next along each process. Events labelled 2 are those that have already been executed.)

The state space of $\mathcal{B}$ is finite because of the bounds B and K that we can compute for a locally synchronized TC-MSG G (Corollary 1). Since every TC-MSC in $L(G)$ is B-bounded, the channel state χ is bounded. Moreover, since the length of the active suffix along any run is at most K, the length of π is bounded, and hence so is ρ.

An initial state of $\mathcal{B}$ is one in which the global state of the timed MPA $\mathcal{A}$ is $\overline{s}_{in} \in \prod_{p\in\mathcal{P}} S_{in}^p$, all channels are empty, η maps each action to 0, the active suffix $\pi = q$ for some initial node $q \in Q_{in}$ and ρ maps the minimal events along each process in $M(q)$ to 1 and all other events to 0.

Clocks of $\mathcal{B}$. Interval constraints in G will be monitored using additional clocks in $\mathcal{B}$. The set of clocks of $\mathcal{B}$ consists of all clocks of $\mathcal{A}$ along with a clock z^{lc} for each constraint lc local to a process in G, a clock z^{EC} for each edge constraint EC in G, and a set of clocks z_i^{mesg}, $1 \leq i \leq B$, one for each potentially active copy of a message constraint $mesg$ in G.

If $lc = \langle (e, e'), I\rangle$ is a local constraint on process p, the clock z^{lc} is reset to zero when $\mathcal{B}$ simulates e and is checked against the interval I when $\mathcal{B}$ simulates e'. Each clock z^{EC} is used to check edge constraints in an analogous manner. For message constraints, we may have multiple copies of the same message in a channel. We have to maintain and check the constraint independently for each copy of the message. However, since channels are B-bounded, there can be no more than B active copies of a message at any point. Thus, we can use the clocks z_i^{mesg} in conjunction with the state information η that assigns a number modulo B to each communication action in order to check message constraints.

Transitions. We can now define the transition function for $\mathcal{B}$. Each move of $\mathcal{B}$ consists of one of the following:

- Pick a process p and perform a legal move in $\mathcal{A}$ that executes the (unique) p-event e_p labelled 1 by ρ. This move is labelled by $\lambda(e_p) \in Act$.
- If there is no active event for some process, extend the active suffix by adding a node q. This move is labelled q.

Transitions on ***Act*****.** Since $\mathcal{B}$ incorporates the global state, the channel state and the clocks of $\mathcal{A}$, it can faithfully simulate every move of $\mathcal{A}$. We use the remaining components of the state of $\mathcal{B}$ to restrict the moves of $\mathcal{A}$ to follow a path in G.

Formally, we have a move $(s, \varphi, \alpha, X, s')$ in $\mathcal{B}$, where φ is a clock constraint, $\alpha \in Act$ and X is a set of clocks to be reset if the following hold. Let us assume that $\alpha \in Act_r$ is an r-action. Then:

- The projection of $(s, \varphi, \alpha, X, s')$ onto components from $\mathcal{A}$ defines a valid (global) transition of $\mathcal{A}$.
- The action α must match the label of the r-event e_r currently labelled 1 by ρ in s.
- If $lc = \langle (e, e_r), I\rangle$ is a local constraint, we must have in φ a constraint checking that z^{lc} satisfies I. For instance, if $I = [m, n)$, we have a constraint $m \leq z^{lc} \wedge z^{lc} < n$ in φ.
- If $EC = \langle (e, e_r), I\rangle$ is an edge constraint, e is the maximum r-event in $M(q_i)$ and e_r is the minimum r-event in $M(q_{i+1})$ for some nodes q_i, q_{i+1} along π, then we must have in φ a constraint checking that z^{EC} satisfies I.
- If $\alpha = r?s(m)$, $\eta(\alpha) = k$ and there is a message constraint $mesg = \langle (e_s, e_r), I\rangle$ from the corresponding send event e_s, we must have in φ a constraint checking that $z^{mesg}_{(k+1) \bmod m}$ satisfies I.

The new state of $\mathcal{B}$, s', is obtained from s as follows.

- The global state of $\mathcal{A}$ is updated according to the transition simulated by $\mathcal{B}$.
- The channel state of $\mathcal{A}$ is updated in the obvious way.
- $\eta(\alpha)$ is updated to $(\eta(\alpha) + 1) \bmod B$.
- $\rho(e_r)$ is set to 2 and the next r event in $M(\pi)$, if any, is labelled 1.
- Let $\pi = q_0 q_1 \ldots q_m$. Let $i_{\max}$ be the maximum index in the set $\{i \mid \forall e \in M(q_i).\ \rho(e) = 2\}$. Update π to $q_{i_{\max}} q_{i_{\max}+1} \ldots q_m$. Note that this deletes all completed nodes from the active suffix except the last one. We need to retain $q_{i_{\max}}$ in the active suffix to check edge constraints.

Finally, we compute the set X of clocks to be reset on this transition as follows:

- If $lc = \langle (e_r, e'), I \rangle$ is a local constraint involving e_r, add z^{lc} to X.
- If e_r is the maximum r-event in $M(q)$, for each edge $q \to q' \in G$, if e' is the minimum r-event in $M(q')$ and $EC = \langle (e_r, e'), I \rangle$ is an edge constraint, add z^{EC} to X.
  Note that we activate clocks for edge constraints along *all* possible extensions of the path when we encounter a maximal event in a node. However, when we reach a minimal event in the next node, we ensure that we only enforce the constraint for the edge that was actually traversed. Each time we traverse an edge with constraint EC, the clock z^{EC} will be reset, so there is no danger when we check an edge constraint that the clock value is stale.
- If $\alpha = r!s(m)$ and there is a message constraint $mesg = \langle (e_r, e_s), I \rangle$ involving e_r, add $z^{EC}_{\eta(\alpha)}$ to X.

Transitions on Q. If $\mathcal{B}$ is in a state $s = (\overline{s}, \chi, \eta, \pi, \rho)$ where for some subset $P \subseteq \mathcal{P}$, for each $p \in P$ there is no p-event labelled 1 by ρ, we need to extend $\pi = q_0 q_1 \ldots q_m$. In such a situation, for each q such that there is an edge $q_m \to q$ in G, we add a transition $s \xrightarrow{q} s'$ in $\mathcal{B}$ where $s' = (\overline{s}, \chi, \eta, \pi \cdot q, \rho')$, such that $\rho'(e)$ agrees with $\rho(e)$ for all $e \in M(\pi)$, $\rho'(e) = 1$ if e is a minimum p-event in $M(q)$ for $p \in P$, and $\rho'(e) = 0$ for all other events $e \in M(q)$.

Accepting states $\mathcal{B}$. A state $s = (\overline{s}, \chi, \eta, \pi, \rho)$ of $\mathcal{B}$ is accepting if $\overline{s}$ is an accepting state of $\mathcal{A}$, $\chi = \chi_\varepsilon$, the channel state in which there are no messages in any channel, $\pi = q_0 q_1 \ldots q_m$ with $q_m \in Q_F$ and $\rho(e) = 2$ for all $e \in M(\pi)$.

From the definition of $\mathcal{B}$, it is routine, though tedious, to prove that $\mathcal{B}$ simulates precisely those runs of $\mathcal{A}$ that cover some path in G. Moreover, for each such run, $\mathcal{B}$ records the sequence of nodes in G traversed along the run. Thus $L_Q(\mathcal{B}) = \pi_Q(\mathit{Untime}(L(\mathcal{B})))$ is a regular language describing the set of paths in G covered by runs of $\mathcal{A}$. As described earlier, we can apply the same construction to the trivial automaton $\mathcal{A}_U$ recognizing Act^* to get a timed automaton $\mathcal{B}_U$ that marks out all feasible paths in G. It then follows that checking coverage is equivalent to checking that $L_Q(\mathcal{B}_U) \subseteq L_Q(\mathcal{B})$.

From Locally Synchronized to Arbitrary TC-MSGs

If we drop the assumption that the TC-MSG we start with is locally synchronized, the automaton $\mathcal{B}$ fails to be finite-state because we cannot guarantee a bound on either the channel capacities or the length of the active suffix.

However, in such a situation, we can still solve a restricted version of the problem. For each MSC of the form $M = M_1 \circ M_2 \circ \cdots \circ M_k \in L(G)$ generated by a path $q_1 q_2 \ldots q_k$ with $M_i = M(q_i)$ for $i \in \{1, 2, \ldots, k\}$, we ask whether there is a timed MSC $T \in \mathcal{L}(\mathcal{A})$ such that $\text{lin}(T) = w_1 w_2 \ldots w_k$ where $w_i \in \text{lin}(M_i)$ for $i \in \{1, 2, \ldots, n\}$. In other words, we restrict our attention to runs of $\mathcal{A}$ that cover G one node at a time.

We can suitably modify the construction of the automaton $\mathcal{B}$ to achieve this. In the new version of $\mathcal{B}$, there is always only one active node, from the way we have defined the simulation. Channel capacities are bounded by the maximum capacity exhibited by the individual TC-MSCs labelling the nodes of G. We can then obtain an analogous result for coverage in terms of $L_Q(\mathcal{B}_U)$ and $L_Q(\mathcal{B})$. Due to lack of space, we omit further details.

Negative Specifications

For negative specifications, the verification problem is dual—given a TC-MSG G and a timed MPA $\mathcal{A}$, we want to ensure that there is no timed MSC $T \in \mathcal{L}(\mathcal{A})$ that covers any TC-MSC $M \in L(G)$. Let us call this problem *avoidance*.

Unlike coverage, avoidance does reduce directly a problem involving timed languages. We want $\text{lin}(L(G))$, the set of timed linearizations of $L(G)$, to be disjoint from $L(\mathcal{A}) = \text{lin}(\mathcal{L}(\mathcal{A}))$, the set of timed linearizations of $\mathcal{L}(\mathcal{A})$. However, there is still the problem of finding an effective presentation of $\text{lin}(L(G))$.

Notice, however, that the construction used to prove Theorem 3 also solves the avoidance problem. Using the terminology introduced above, avoidance is equivalent to checking that $L_Q(\mathcal{B}_U) \cap L_Q(\mathcal{B}) = \emptyset$.

7 Discussion

We have shown how to solve the timed analogue of the scenario matching problem for specifications consisting of infinite sets of time-constrained MSCs. Though our result is stated in the context of timed MPAs, our construction works even if the system is presented as a global timed automaton. Our solution to the coverage problem for positive specifications also yields a solution to the avoidance problem for negative specifications. An interesting problem to be tackled is realizability for time-constrained MSGs—given a TC-MSG G, construct a timed MPA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A})$ covers $L(G)$.

References

1. Alur, R., Dill, D.: A Theory of Timed Automata. Theor. Comput. Sci. 126, 183–225 (1994)
2. Alur, R., Yannakakis, M.: Model checking of message sequence charts. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 114–129. Springer, Heidelberg (1999)
3. Bengtsson, J., Yi, W.: Timed Automata: Semantics, Algorithms and Tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)
4. Chandrasekaran, P., Mukund, M.: Matching Scenarios with Timing Constraints. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 98–112. Springer, Heidelberg (2006)

5. D'Souza, D., Mukund, M.: Checking consistency of SDL+MSC specifications. In: Ball, T., Rajamani, S.K. (eds.) Model Checking Software. LNCS, vol. 2648, pp. 151–165. Springer, Heidelberg (2003)
6. Henriksen, J.G., Mukund, M., Narayan Kumar, K., Sohoni, M., Thiagarajan, P.S.: A Theory of Regular MSC Languages. Inf. Comp. 202(1), 1–38 (2005)
7. ITU-T Recommendation Z.120: Message Sequence Chart (MSC). ITU, Geneva (1999)
8. Mauw, S., Reniers, M.A.: High-level message sequence charts. In: Proc. SDL'97, pp. 291–306. Elsevier, Amsterdam (1997)
9. Muscholl, A., Peled, D.: Message sequence graphs and decision problems on Mazurkiewicz traces. In: Kutyłowski, M., Wierzbicki, T., Pacholski, L. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 81–91. Springer, Heidelberg (1999)
10. Muscholl, A., Peled, D., Su, Z.: Deciding properties for message sequence charts. In: Nivat, M. (ed.) ETAPS 1998 and FOSSACS 1998. LNCS, vol. 1378, pp. 226–242. Springer, Heidelberg (1998)

Is Observational Congruence Axiomatisable in Equational Horn Logic?

Michael Mendler[1,⋆] and Gerald Lüttgen[2]

[1] Informatics Theory Group, University of Bamberg, Germany
michael.mendler@wiai.uni-bamberg.de
[2] Department of Computer Science, University of York, UK
gerald.luettgen@cs.york.ac.uk

Abstract. It is well known that bisimulation on μ-expressions cannot be finitely axiomatised in equational logic. Complete axiomatisations such as those of Milner and Bloom/Ésik necessarily involve implicational rules. However, both systems rely on features which go beyond pure equational Horn logic: either the rules are impure by involving non-equational side-conditions, or they are schematically infinitary like the congruence rule which is not Horn. It is an open question whether these complications cannot be avoided in the proof-theoretically and computationally clean and powerful setting of second-order equational Horn logic.

This paper presents a positive and a negative result regarding axiomatisability of observational congruence in equational Horn logic. Firstly, we show how Milner's impure rule system can be reworked into a pure Horn axiomatisation that is complete for guarded processes. Secondly, we prove that for unguarded processes, both Milner's and Bloom/Ésik's axiomatisations are incomplete without the congruence rule, and neither system has a complete extension in rank 1 equational axioms. It remains open whether there are higher-rank equational axioms or Horn rules which would render Milner's or Bloom/Ésik's axiomatisations complete for unguarded processes.

1 Introduction

The existence and nonexistence of equational axiomatisations of behavioural equivalences in process algebra has received significant interest in the literature [2,8,23,24,26]. Most recent work is concerned with finite processes and equational axiomatisations for a range of operators (such as for priority [1]) and behavioural semantics (such as for simulation equivalence [9]). The focus on finite processes is natural since many behavioural relations cannot be finitely axiomatised in the presence of recursion. This has long been known for regular expressions [11] and was shown to apply to μ-expressions as well [8,26]. Except for special and not very well understood situations in the language of $*$-expressions [11,13,14], purely equational theories appear to be inadequate for

⋆ The author acknowledges support by the EU Types network IST 510996.

recursive processes. Thus, a more powerful setting is needed in order to study the relative proof-theoretic complexities of theories for regular processes.

A suitable and quite natural setting is provided by *(second-order) equational Horn logic* [25]. Indeed, Milner's axiomatisation of strong bisimulation for finite state processes [20] and Bloom/Ésik's abstract generalisation [6,12] involve *conditional equations*, as does Milner's axiomatisation of *observational congruence* [22] and Glabbeek's axiomatisation of branching bisimulation [16], or the various bisimulation-style equivalences in timed process algebras [3,4,5,10]. Looking at these in detail, however, reveals that they are not strictly Horn theories because they depend on the congruence rule for recursion (cf. rule C4 below) which is not Horn, and they are not pure because rules have guardedness side conditions (cf. rule R2 below).

$$\mathsf{C4}\quad \frac{E = F}{\mu x.\, E = \mu x.\, F} \qquad\qquad \mathsf{R2}\quad \frac{F = E\{F/x\}}{\mu x.\, E = F}\; x \text{ guarded in } E$$

To see that rule C4 is not Horn, consider the soundness of C4 which logically corresponds to the formula $(\forall x.\, E = F) \supset \mu x.\, E = \mu x.\, F$. This formula is not Horn since the precondition of the implication is universally quantified; in [12], this is called an *implication between equations*. The Horn interpretation of C4 would be $\forall x.\ (E = F \supset \mu x.\, E = \mu x.\, F)$ which is unsound. Take for example $E \equiv a.x$ and $F \equiv a.\mathsf{0}$. Then, the equation $E\{\mathsf{0}/x\} = F\{\mathsf{0}/x\}$ is sound, but $\mu x.\, a.x$ is not bisimilar to $\mu x.\, a.\mathsf{0}$. In the Horn theory of closed terms, rule C4 is only *admissible* in the sense that, if all closed instantiations of $E = F$ are derivable, then all closed instantiations of $\mu x.\, E = \mu x.\, F$ are derivable, too. But this rule is infinitary and not expressible in Horn form. This leads us to the following – in our opinion – key open problem:

> *Can bisimulation for finite state processes be axiomatised in pure equational Horn logic?*

The answer to this question relates to the issue of guardedness. On the face of it, C4 appears to be necessary to prove equalities between recursive processes. Consider processes $p =_{\mathrm{df}} \mu x.\, (\alpha.x + \beta.x)$ and $q =_{\mathrm{df}} \mu x.\, (\beta.x + \alpha.x)$, where "$\alpha$." and "$\beta$." are action prefixes, x is a process variable, "μx." the recursion operator and "+" non-deterministic choice. The processes p and q are bisimilar, and the equation $p = q$ can be derived by first applying the commutativity law on open terms $\alpha.x + \beta.x = \beta.x + \alpha.x$ and then closing under recursion using C4. Interestingly, for guarded processes, i.e., if both α and β are observable actions, the same is achieved without C4. Using recursive unfolding and commutativity, one derives $p = \alpha.p + \beta.p$ and $q = \alpha.q + \beta.q$, i.e., both p and q provably satisfy the same guarded equation system. From there, by way of rule R2, symmetry and transitivity of equality, one finally gets $p = \mu x.(\alpha.x + \beta.x) = q$.

Due to this issue of unguardedness, the above question is particularly challenging for *observational congruence* [21]. The question's importance lies in the fact that the Horn rule format is crucial for standard automated reasoning based on Prolog-style SLD resolution. Moreover, the question is an interesting one since, as we will show, Bloom and Ésik's axiomatisation which is commonly considered pure Horn is in fact not Horn, and Milner's axiomatisation which is commonly considered impure is in fact pure, i.e., guardedness is equational.

As our first technical result, we provide an axiomatisation of observational congruence for finite state processes which is in pure equational Horn form. This axiomatisation is an adaptation of Milner's proof system and interprets the underlying equality as *partial equivalence* via which we may encode the side condition of rule $\mathsf{R2}$. Our axiom system is sound for all processes and complete for guarded processes. Hence, the question remains whether this axiom system can be extended to handle unguardedness. As our second technical result, we show that no finite rank 1 *equational* extension of Milner's axiom system yields completeness for unguarded processes, not even when including the impure rule $\mathsf{R2}$ or the pure *GA-implication* rule of Bloom/Ésik [26]. To the best of our knowledge, this result is the first negative result on process-algebraic axiomatisations in equational Horn logic to be reported in the literature. It can be generalised to rank 2 and provides a number of technical insights into the proof-theoretic expressiveness of Horn logic for observational congruence. Specifically, we conjecture that unguardedness on μ-expressions cannot be axiomatised in second-order equational Horn logic of any rank. Note that for $*$-expressions this problem does not occur. In [18], Kozen presented a finitary axiomatisation of the Kleene algebra of $*$-expressions involving only pure equational implications. Since $*$-expressions do not have an explicit recursion binder, a congruence rule like $\mathsf{C4}$ is therefore not needed in Kleene algebra. Finally, note that the proofs of our results can be found in a technical report [19].

2 The Process Language μBCCSP^2

Variable-binding operators require second-order matching, in order to handle syntactic contexts such as the bodies F of recursive processes $\mu x. F$. This section introduces our process language and makes precise what we understand by a second-order Horn axiomatisation. In particular, the language must be general enough to capture not only the object-level syntax of processes but also the meta-level syntax of schemes and rules needed to formalise logic deduction. Our language μBCCSP^2 is an extension of BCCSP [15] by recursion and schematic variables. It corresponds to the second-order fragment T^2 of [26].

Second-Order Syntax, Semantics and Observational Congruence. The second-order language of (schematic, context) *μ-expressions*, or *expressions* for short, is defined by

$$F \quad ::= \quad x(F_1, F_2, \ldots, F_n) \mid \$\mathsf{k} \mid \mathsf{0} \mid \alpha.F \mid F_1 + F_2 \mid \mu x. F\,.$$

It includes *variables* x and the usual process-algebraic operators of *prefixing* $\alpha.F$, *summation* $F_1 + F_2$ and *recursion* $\mu x. F$. The *prefixes* α range over a denumerable set of *observable actions* $a_0, a_1, a_2, \ldots$ and the distinguished *silent action* τ. The constant $\mathsf{0}$ represents the *inactive* process. The expressions $\$\mathsf{k}$ are *call-back constants*, where $k \geq 1$, which will be used to form contexts.

Every variable x has a *(context) rank* which specifies the number of parameters that x must be instantiated with to form a process. This is done in *(context) applications* of the form $x(F_1, F_2, \ldots, F_n)$, where $\mathit{rank}(x) = n$. We assume that there is a countably infinite number of variables at every rank. Rank 0 variables are called *process variables* and all other variables *schematic variables*. For process variables we simply write x instead of $x()$. Recursion is possible over process variables only, i.e., we require $\mathit{rank}(x) = 0$ in any expression $\mu x.\, F$. The variable x in $x(F_1, F_2, \ldots, F_n)$ stands for a context with uniquely identified syntactic slots into which the expressions F_i, for $1 \leq i \leq n$, are inserted. These slots are represented by the call-back constants $\$1, \$2, \ldots, \$n$. Formally speaking, call-back constants are nothing but implicitly bound and canonically named process variables. These would be represented as explicit λ-abstractions in higher-order systems like [26]. The result of *instantiating* x by expression F is written $F[F_1, F_2, \ldots, F_n]$ and obtained if each occurrence of $\mathsf{\$k}$ in F is substituted by F_k. We say that F has *rank* n if it does not contain call-back constants larger than $\mathsf{\$n}$. Expressions of rank 0 are called *process schemes*, and those of higher rank are called *contexts* or *expressions*. Thus, if F has rank n and all F_i, for $1 \leq i \leq n$, are process schemes, then $F[F_1, F_2, \ldots, F_n]$ is a process scheme.

The recursion operator $\mu x.\, F$ binds all occurrences of process variable x in F. There is no variable binder for schematic variables. The notions of *free* and *bound* occurrences of variables and of *guardedness* of variables are as usual. In particular, a variable x is called guarded in an expression F, if all occurrences of x in F are within the scope of an α-prefix with $\alpha \not= \tau$. An expression F without free variables (of any rank) is *closed*; otherwise it is *open*. Process schemes without schematic variables, i.e., both rank and variable rank are 0, are called *process terms*. Process terms without free process variables are *process constants*, or simply *processes*. We use $E, F, \ldots$ to range over general expressions, $t, u, \ldots$ to range over process terms, and $p, q, \ldots$ for process constants. We let $\equiv$ stand for the syntactic identity on expressions and denote the sub-expression relation by $\trianglelefteq$, i.e., $E \trianglelefteq F$ if either E is a proper sub-expression of F or if $E \equiv F$. Besides the meta-level identity $E \equiv F$ on expressions we consider formal equalities $E = F$ between process schemes, called *equation schemes*. By the *rank* of an equation scheme $E = F$ we understand the maximal variable rank of E and F. As noted above, the rank of a variable specifies the rank of the context expression by which it needs to be instantiated to generate a process scheme.

An *instantiation* σ is a finite partial mapping from variables to expressions which is rank-preserving, i.e., such that for any variable x in the *domain* of σ, expression $\sigma(x)$ is of rank $\mathit{rank}(x)$. If E is an expression with free variables X and σ an instantiation with domain X, the *instantiation of E by σ*, written $\sigma(E)$, is obtained by recursively replacing each sub-expression $x(F_1, F_2, \ldots, F_n) \trianglelefteq E$ by $\sigma(x)[\sigma(F_1), \sigma(F_2), \ldots, \sigma(F_n)]$. This is a second-order operation which is to be distinguished from the standard first-order *substitution* $E\{F/x\}$ in which variable x is replaced by F in a single recursive pass through E. For instance, if x and y are two variables of rank 0 and 1, respectively, and $E =_{\mathrm{df}} x(y)$, then the substitution σ with $\sigma(x) =_{\mathrm{df}} y + \mathsf{\$1}$, $\sigma(y) =_{\mathrm{df}} \mathsf{0}$ yields $\sigma(E) \equiv \sigma(x(y)) \equiv \sigma(x)[\sigma(y)] \equiv (y + \mathsf{\$1})[\mathsf{0}] \equiv y + \mathsf{0}$, while substitution $E\{y + \mathsf{\$1}/x\}\{\mathsf{0}/y\}$ would return $(y + \mathsf{\$1})(\mathsf{0})$

which is not well-formed. Instantiations preserve well-formedness and rank. In particular, if E is a process scheme, then $\sigma(E)$ is again a process scheme.

Preserving well-formedness is not enough for instantiations to be sensible in equational reasoning for recursive processes with variable binding. It must be ensured that in the instantiation $\sigma(E) = \sigma(F)$ of an equation $E = F$ we do not inadvertently capture free process variables inside E or F. An instantiation σ is called *free* for E, if its application $\sigma(E)$ avoids name capture of free process variables, i.e., every occurrence of a free variable in $\sigma(x)$ remains free after instantiation into $\sigma(E)$. We will use symbol θ to range over free instantiations. In practise, there are two options to keep instantiations free. One is to require that θ is *closed*, i.e., for all x in its domain, $\theta(x)$ is closed. The other is to rename bound variables systematically, e.g., by taking expressions up to α-conversion.

The semantics of μBCCSP^2 is the transition system induced by process constants as states and where the action–labelled transition relation is inductively defined by the standard operational rules:

$$\frac{—}{\alpha.\, p \xrightarrow{\alpha} p} \qquad \frac{p_1 \xrightarrow{\alpha} q}{p_1 + p_2 \xrightarrow{\alpha} q} \qquad \frac{p_2 \xrightarrow{\alpha} q}{p_1 + p_2 \xrightarrow{\alpha} q} \qquad \frac{t\{\mu x.\, t/x\} \xrightarrow{\alpha} q}{\mu x.\, t \xrightarrow{\alpha} q}$$

Finally, recall the definition of *observational equivalence* and *observational congruence* [21]. As usual, $\overset{\epsilon}{\Longrightarrow}$ stands for $(\xrightarrow{\tau})^*$, $\overset{\alpha}{\Longrightarrow}$ denotes $\overset{\epsilon}{\Longrightarrow} \circ \xrightarrow{\alpha} \circ \overset{\epsilon}{\Longrightarrow}$, and $\hat{\alpha} =_{\mathrm{df}} \alpha$, if $\alpha \neq \tau$, and $\hat{\tau} =_{\mathrm{df}} \epsilon$. A symmetric binary relation $\mathcal{R}$ on process constants is a *weak bisimulation relation* if

$$\forall \langle p, q\rangle \in \mathcal{R}.\ \forall \alpha, p'.\ (p \xrightarrow{\alpha} p' \text{ implies } \exists q'.\, q \overset{\hat{\alpha}}{\Longrightarrow} q' \text{ and } \langle p', q'\rangle \in \mathcal{R}).$$

The largest such relation $\approx$ is an equivalence and referred to as *observational equivalence*. The largest congruence $\approxeq$ contained in $\approx$, called *observational congruence*, is characterised by the condition that $p \approxeq q$ iff

$$\forall \alpha, p'.\ (p \xrightarrow{\alpha} p' \text{ implies } \exists q'.\, q \overset{\alpha}{\Longrightarrow} q' \text{ and } p' \approx q'),$$

and symmetrically. The relation $\approxeq$ is lifted to process schemes E, F by universal abstraction: $E \approxeq F$, if $\theta(E) \approxeq \theta(F)$ for all closed instantiations θ.

Second-Order Equational Horn Logic. If E and F are two well-formed process schemes, then formal equations $E = F$ are of second order, also known as *hyper-identities* [8]. This is because of the presence of schematic variables in our setting. A (pure) second-order equational Horn system is a finite set of *Horn rules*, i.e., rules of the form

$$\frac{E_1 = F_1 \quad \cdots \quad E_n = F_n}{E = F} \ ,$$

where the $E_i = F_i$ are referred to as the rule's *premises* and $E = F$ as the rule's *conclusion*. If the rule has no premises, i.e., $n = 0$, then it is called *axiom*. Given a finite set $\mathcal{A}$ of Horn rules, we say that an equation scheme $G = H$ is derivable from $\mathcal{A}$, in symbols $\mathcal{A} \vdash G = H$, if there exists a finite sequence of equation schemes $G_0 = H_0,\ G_1 = H_1,\ \ldots,\ G_n = H_n$ such that (a) $G \equiv G_n$ and $H \equiv H_n$; and (b) every equation $G_i = H_i$ is derived by instantiating some Horn rule

$$\frac{E_1 = F_1 \quad \cdots \quad E_m = F_m}{E = F}$$

from $\mathcal{A}$ by way of a free instantiation θ_i such that (i) $\theta_i(E) \equiv G_i$, $\theta_i(F) \equiv H_i$ and (ii) for all $1 \leq s \leq m$ there exists an index $r < i$ satisfying $\theta_i(E_s) \equiv G_r$ and $\theta_i(F_s) \equiv H_r$. Permitting arbitrary free instantiations yields a rather general notion of deduction for Horn theories. In particular, we can derive equations $\mathcal{A} \vdash t = u$ between open process terms.

Naturally, a theory $\mathcal{A}$ is sound if $\mathcal{A} \vdash G = H$ implies $G \approxeq H$, i.e., $\theta(G) \approxeq \theta(H)$ for all closed instantiations θ. For this to hold true, each Horn rule must be sound in the sense that for all closed instantiations θ, if $\forall i.\, \theta(E_i) \approxeq \theta(F_i)$, then $\theta(E) \approxeq \theta(F)$. This interpretation of soundness, where the universal quantifier over the interpretation of free variables covers the whole rule, is the definitive characteristic of Horn logic. It is important to note that this is something very different from the implication $(\forall i.\, E_i \approxeq F_i) \supset E \approxeq F$, which would be saying that for all closed θ, $\theta(E) \approxeq \theta(F)$ if for all closed θ and i, $\theta(E_i) \approxeq \theta(F_i)$. This is a strictly weaker soundness criterion. The former and stronger Horn-style soundness is the basis for the standard process of Prolog-style SLD resolution, which is known to be complete for Horn theories and ground goals. On open goals $G = H$, the backward proof search generates closed solution instantiations through unification, essentially treating the free variables in the goal as existential or *flexible*. That this works is due to the strong soundness of Horn rules. The difference from the usual first-order setting is that we permit instantiation of schemes by syntactic context functions, which requires *second-order unification*. We refer the reader to [25] for more details on higher-order unification and the proof theory of higher-order Horn logic.

3 A Pure Horn Axiomatisation

This section shows that the side condition "x guarded in E" in Milner's rule R2 can be eliminated in pure equational Horn logic. The key idea is to re-interpret equations so that they only relate *extensional* processes.

The syntactic relation $\unrhd$ of *weak visibility* is the least relation which satisfies the rules $t \unrhd t$ and, if $t \unrhd r$, then $t + u \unrhd r$, $u + t \unrhd r$, $\tau.t \unrhd r$ and $\mu y.\, t \unrhd r\{\mu y.\, t/y\}$. Intuitively, $t \unrhd r$ states that r occurs weakly unguarded in t. Note that $\unrhd$ abstracts from τ-actions unlike the strong form of $\rhd$ in [22,26]. A process term t is called *extensional* if there is no term $\mu y.\, u$ such that $t \unrhd \mu y.\, u$ and $u \unrhd y$. Hence, an extensional process term is a process term that cannot engage in an initial divergence. For example, process $\mu x.\, (a.x + b.x)$ is extensional whereas $\mu x.\, (a.x + \tau.x)$, $\tau.\mu x.\, (a.x + \tau.x)$ and $\mu x.\, x$ are not. Moreover, every guarded process is extensional, but not vice versa, e.g., $a.\mu x.\, x$ is extensional but not guarded. On the other hand, whenever $\mu x.t$ is extensional, x is guarded in t.

We now provide a sound axiomatisation of $\approxeq$ restricted to extensional processes. This is only a partial equivalence relation, i.e., a relation that is transitive and symmetric but not reflexive. It turns out that with this modification, the side condition of Milner's rule R2 can be expressed purely equationally. In the following, we reconstruct Milner's original axiomatisation of observational congruence as a *pure* equational Horn theory. For notational convenience, we abbreviate the reflexive equation $E = E$ by $E\downarrow$. To begin with, any algebraic axiomatisation depends on reflexivity, symmetry, transitivity and congruence of equality, all of which may be cast into Horn rules:

$$\mathsf{Eq1}\ \frac{-}{\mathsf{0}\downarrow} \qquad \mathsf{Eq2}\ \frac{-}{a.x\downarrow} \qquad \mathsf{Eq3}\ \frac{z(\mu x.\, z(x))\downarrow}{\mu x.\, z(x)\downarrow} \qquad \mathsf{Eq4}\ \frac{x = y}{y = x}$$

$$\mathsf{Eq5}\ \frac{x = y \quad y = z}{x = z} \qquad \mathsf{C1}\ \frac{x = y}{\alpha.x = \alpha.y} \qquad \mathsf{C2}\ \frac{x_1 = y_1 \quad x_2 = y_2}{x_1 + x_2 = y_1 + y_2}$$

Here, all x, x_i, y, y_i are process variables, z is a schematic variable of rank 1, α is an arbitrary action and a stands for an action different from τ. (Strictly speaking, for finite axiomatisation, a, α must be read as a special form of action variables.) Eq1–Eq3 are reflexivity rules. Together with Eq4, Eq5, C1 and C2, the above rules yield a weak extension of the standard equational theory for finite processes by recursion. It is weak since it proves $p = p$ for extensional processes only, as shown in Prop. 1 below.

The standard equational axioms of commutativity, associativity, idempotence and neutrality, as well as Milner's τ-laws can be phrased as Horn rules, too:

$$\mathsf{S1}\ \frac{x_1 + x_2 = y}{x_2 + x_1 = y} \quad \mathsf{S2}\ \frac{(x_1 + x_2) + x_3 = y}{x_1 + (x_2 + x_3) = y} \quad \mathsf{S3}\ \frac{x = y}{x + y = x} \quad \mathsf{S4}\ \frac{x = y}{x + \mathsf{0} = y}$$

$$\mathsf{T1}\ \frac{\alpha.x = y}{\alpha.\tau.x = y} \quad \mathsf{T2}\ \frac{\tau.x = y}{x + \tau.x = y} \quad \mathsf{T3}\ \frac{\alpha.(x_1 + \tau.x_2) = y}{\alpha.x_2 + \alpha.(x_1 + \tau.x_2) = y}$$

Finally, consider the following rules for the recursion operator, both of which are variations of Milner's recursion rules [22]:

$$\mathsf{R1}\ \frac{\mu x.\, z(x) = y}{z(\mu x.\, z(x)) = y} \qquad \mathsf{R2}^*\ \frac{x = z(x) \qquad \mu x.\, z(x) = y}{x = y}$$

R1 expresses that $\mu x.\, t$ is a solution of the fixed point equation $x = t$. This is usually represented by an equation scheme $\mu x.\, t = t\{\mu x.\, t/x\}$ for the direct unfolding of recursive processes. Our formulation uses a conditional form which essentially restricts the recursive unfolding to the cases where $\mu x.\, t$ is extensional. The second rule R2* states that extensional equations have unique recursive solutions. More precisely, if $p = t\{p/x\}$ and if the fixed point $\mu x.\, t$ is provably identical to some process q, then p and q are identical. Hence, if $\mu x.\, t$ is extensional, then all solutions of the equation $x = t$ are equal to $\mu x.\, t$. The second premise $\mu x.\, z(x) = y$ of R2* takes the place of the non-equational side condition "x guarded in E" in Milner's R2, in the sense that $\mu x.\, z(x) = y$ can only be derived if z is a guarded context.

Let M* be the system of axioms Eq1–Eq5, C1–C2, S1–S4, T1–T3, R1, R2*. Observe that M* is a pure equational Horn axiomatisation in rank 1.

Proposition 1. *A process p is extensional iff $\mathsf{M}^* \vdash p\downarrow$.*

It is important to note that the statement of Prop. 1 is non-monotonic in the number of axioms. Adding axioms to M^* may yield provable reflexivities for non-extensional processes, while removing axioms may mean that some extensional processes are not verifiably reflexive any longer.

Theorem 1. *M^* is sound regarding $\approxeq$ for all processes and complete for guarded processes.*

Proof. The proof is a replay of Milner's proof [22]. For soundness one observes that the side condition of Milner's rule $\mathsf{R2}$ is captured by the equation $\mu x.\, z(x) = y$ in $\mathsf{R2}^*$. For if $\mathsf{M}^* \vdash \mu x.\, E = p$, then by symmetry and transitivity $\mathsf{M}^* \vdash (\mu x.\, E) \downarrow$, which means that $\mu x.\, E$ is extensional by Prop. 1 and thus x is guarded in E. For completeness one observes that, by Prop. 1 and since guardedness implies extensionality, $\mathsf{M}^* \vdash p \downarrow$ is derivable for every guarded process, and also that all rules of Milner can be simulated by the associated rule in M^*. □

Thm. 1 implies that the deductive mechanism of equational (second-order) Horn logic is sufficient to axiomatise recursion on the fragment of (closed) processes. The salient feature of Milner's proof was to show that the infinitary nature of rule $\mathsf{C4}$ can be localised completely in the question of guardedness. Our result shows that the guardedness side condition can be captured equationally in the form of extensionality. Thus, Milner's rank 1 axiomatisation – counter to common belief – is essentially pure Horn. The restriction of completeness to *guarded* processes does not affect expressiveness. Many process algebras (see, e.g., [5]) and tools are based on guarded recursive specifications, and it is well known that every unguarded process is provably equivalent to a guarded one [22]. However, as we shall see next, this latter property seems to depend crucially on the presence of non-Horn rule $\mathsf{C4}$ which is implicit in Milner's original article [22].

4 Can Horn Eliminate Unguardedness?

As seen above, observational congruence of guarded processes in μBCCSP^2 can indeed be formalised in pure equational Horn logic of (variable) rank 1. We now show that for general, unguarded processes, neither Milner's axiomatisation [22] nor Bloom/Ésik's axiomatisation (see Sewell [26]) are complete when leaving out the only non-Horn rule $\mathsf{C4}$. Moreover, both cannot be made complete by adding a finite number of rank 1 equational axioms. We first establish our incompleteness result considering equational axioms, and then lift it to include the standard recursion rules employed by Milner and Bloom/Ésik.

Our plan is to show that certain sound equations cannot be derived from any finite and sound equational axiomatisation of $\approxeq$. These equations involve choices between pairwise distinct actions a_i, for $0 \leq i \leq n-1$, where $n \in \mathbb{N}$. Such a choice can be written in a straightforward way, say as the process $\mathbf{A}_n =_{\mathrm{df}} \tau.\, \sum_{i=0}^{n-1} a_i.\mathsf{0}$, or expressed in a more complex manner through a recursive maze of τ-transitions, each of which postpones the choice without preempting any of the actions a_i.

A special class of such terms are constructed from the following family E^n_k of context expressions, indexed by $k \geq 0$ and $n \geq \max(k, 1)$:

$$\begin{aligned}
E^1_0 &=_{\mathrm{df}} \$1 \\
E^{i+2}_0 &=_{\mathrm{df}} \$1 + E^{i+1}_0[\$2, \ldots, \$(\mathsf{i}{+}2)] \\
E^{i+1}_{j+1} &=_{\mathrm{df}} \mu x_{i-j}.(\$1 + \tau.\, E^{i+1}_j[\$2, \$3, \ldots, \$(\mathsf{i}{+}1), x_{i-j}]),
\end{aligned}$$

where $x_0, x_1, \ldots, x_n$ are pairwise distinct process variables. Each E^n_k is closed and of rank n with k bound variables $x_{n-k}, x_{n-k+1}, \ldots, x_{n-1}$, for instance:

$$\begin{aligned}
E^3_3 &\equiv \mu x_0.\, (\$1 + \tau.\, E^3_2[\$2, \$3, x_0]) \\
&\equiv \mu x_0.\, (\$1 + \tau.\, \mu x_1.\, (\$2 + \tau.\, E^3_1[\$3, x_0, x_1])) \\
&\equiv \mu x_0.\, (\$1 + \tau.\, \mu x_1.\, (\$2 + \tau.\, \mu x_2.\, (\$3 + \tau.\, E^3_0[x_0, x_1, x_2]))) \\
&\equiv \mu x_0.\, (\$1 + \tau.\, \mu x_1.\, (\$2 + \tau.\, \mu x_2.\, (\$3 + \tau.\, (x_0 + (x_1 + x_2))))).
\end{aligned}$$

If $\tilde{a}^n_0$ is a shorthand for the sequence $\tilde{a} =_{\mathrm{df}} a_0.\mathsf{0}, a_1.\mathsf{0}, \ldots, a_{n-1}.\mathsf{0}$, then $E^n_n[\tilde{a}] \approxeq \mathbf{A}_n$. However, as we will see, no finite (rank 1) axiomatisation can derive $E^n_n[\tilde{a}] = \mathbf{A}_n$ for every n. The reason is that the syntactic structure of the E^n_k is judiciously chosen in such a way that they behave atomically under second-order syntactic matching. More specifically, in every solution of an equation $w(\tilde{y}) = E^n_k[\tilde{z}]$ for rank m variable w and process variables $\tilde{y} = y_1, y_2, \ldots, y_m$ and $\tilde{z} = z_0, z_1, \ldots, z_{n-1}$, the context E^n_k must either be contained wholesale in w or in some y_i, rather than be split across w and $\tilde{y}$.

Proposition 2. *Let θ be a free instantiation such that $\theta(w)[\theta(\tilde{y})] \equiv E^n_k[\tilde{U}]$ for rank m variable w, process variables $\tilde{y} = y_1, y_2, \ldots, y_m$ and schemes $\tilde{U} = U_0, U_1, \ldots, U_{n-1}$. Then, either $\exists i.\, \theta(y_i) \equiv E^n_k[\tilde{U}]$, or $\exists$ rank m contexts $\tilde{V} = V_0, V_1, \ldots, V_{n-1}$ such that $\theta(w) \equiv E^n_k[\tilde{V}]$ and $V_i[\theta(\tilde{y})] \equiv U_i$, for $0 \leq i \leq n-1$.*

As an example, consider how the expression $E^3_2[a_1.\mathsf{0}, a_2.\mathsf{0}, x_0] \equiv \mu x_1.\, (a_1.\mathsf{0} + \tau.\, \mu x_2.\, (a_2.\mathsf{0} + \tau.\, (x_0 + (x_1 + x_2))))$ may be matched against the pattern $w(\tilde{y})$ with some instantiation θ. Recalling $E^3_2[a_1.\mathsf{0}, a_2.\mathsf{0}, x_0] \trianglelefteq E^3_3[\tilde{a}]$, let us further assume that θ is free for $E^3_3[\tilde{a}]$, i.e., $\theta(w)$ must not have variable x_0 free. This means that $E^3_2[a_1.\mathsf{0}, a_2.\mathsf{0}, x_0]$ cannot be generated from a rank 0 pattern w. In rank 1 there is exactly one nontrivial solution to match the pattern $w(y)$, namely $\theta(w) =_{\mathrm{df}} E^3_2[a_1.\mathsf{0}, a_2.\mathsf{0}, \$1]$ and $\theta(y) =_{\mathrm{df}} x_0$. Here, 'nontrivial' means that $\theta(w)$ uses at least one call-back but is not identical to it. There are more possibilities for the rank 2 pattern $w(y_1, y_2)$. One of these is $\theta(w) =_{\mathrm{df}} E^3_2[a_1.\mathsf{0}, \$1, \$2]$, $\theta(y_1) =_{\mathrm{df}} a_2.\mathsf{0}$ and $\theta(y_2) =_{\mathrm{df}} x_0$. Another one is $\theta(w) =_{\mathrm{df}} E^3_2[a_1.\$2, a_2.\, \$2, \$1]$, $\theta(y_1) =_{\mathrm{df}} x_0$ and $\theta(y_2) =_{\mathrm{df}} \mathsf{0}$. The picture is similar for rank 3 pattern $w(y_1, y_2, y_3)$, in the sense that the context E^3_2 is never broken and the call-back arguments $\theta(y_j)$ generate sub-expressions of $a_i.\, \mathsf{0}$ with the only constraint that one of $\theta(y_i)$ must be identical to x_0. Now take a look at $E^3_1[a_2.\mathsf{0}, x_0, x_1] \trianglelefteq E^3_3[\tilde{a}]$. This time, if θ is to be free for $E^3_3[\tilde{a}]$ again, there is no way in which $E^3_1[a_2.\mathsf{0}, x_0, x_1]$ can match the rank 1 pattern $w(y)$. Both variables x_0 and x_1 would have to be introduced by the call-back $\theta(y)$, which is not possible. On the other hand, in rank 2 against pattern $w(y_1, y_2)$ we can find a match by setting $\theta(w) =_{\mathrm{df}} \mu x_2.\, (a_2.\mathsf{0} + \tau.\, (\$1 + (\$2 + x_2)))$, $\theta(y_1) =_{\mathrm{df}} x_0$ and $\theta(y_2) = x_1$.

The importance of Prop. 2 is that, if we restrict a scheme G to have at most rank m variables, then in any matching $\theta(G) \equiv E_n^n[\tilde{U}]$ all the contexts E_k^n for $n-km$ must either be fully contained in G or fully instantiated via θ from variables in G. For example, if we match $E_3^3[\tilde{a}]$ against a scheme G to find an instantiation θ (free for G) so that $\theta(G) \equiv E_3^3[\tilde{a}]$ then, depending on G either the context E_3^3 is fully contained in G or some context E_k^3, for $0 \leq k \leq 3$, is fully introduced by θ. An example of the first kind would be $G =_{\mathrm{df}} E_3^3[y_0, a_1.\, y_1, a_2.\, y_1]$, $\theta(y_0) =_{\mathrm{df}} a_0.\mathsf{0}$ and $\theta(y_1) =_{\mathrm{df}} \mathsf{0}$. Because of what has been discussed above, if E_k^3, for $k = 1, 2, 3$, is to be introduced by θ, we need a variable of rank at least $3 - k$. For instance, $G =_{\mathrm{df}} y$ and $\theta(y) =_{\mathrm{df}} E_3^3[\tilde{a}]$ would introduce E_3^3 wholesale using a rank 0 variable. Further, $G =_{\mathrm{df}} \mu x_0.\, (a_0.\mathsf{0} + \tau.w(x_0))$ and $\theta(w) =_{\mathrm{df}} E_2^3[a_1.\mathsf{0}, a_2.\mathsf{0}, \$1]$, where w has rank 1, or $G =_{\mathrm{df}} \mu x_0.\, (a_0.\mathsf{0} + \tau.\, \mu x_1.\, w(x_1, x_0))$ with $\theta(w) =_{\mathrm{df}} a_1.\mathsf{0} + \tau.\, E_1^3[a_2.\mathsf{0}, \$2, \$1]$ for rank 2, are solutions introducing E_2^3 and E_1^3, respectively, through θ. In other words, under rank restriction, the E_k^n behave atomically with respect to second-order matching. In this paper we shall explore this feature of the indecomposable expressions E_{n-1}^n to prove non-axiomatisability when using only rank 1 schemes. We believe that the families of expressions E_{n-m}^n can be adapted for obtaining non-axiomatisability with respect to maximal rank m, but leave this to future work.

To obtain our negative results we must generalise the processes $E_n^n[\tilde{a}] \approxeq \mathbf{A}_n$ so that they become robust against attempts to transform them under equational reasoning for $\approxeq$. This means that we need to express their essential structural property in slightly more abstract terms. To this end, let Z be a set of variables of rank n and ξ_n^Z the instantiation with domain Z satisfying $\xi_n^Z(z) = E_{n-1}^n$, where $n = \mathit{rank}(z)$. An expression P is called Z-*pure* if P is of rank 0, i.e., a process scheme, and if it does not contain any variables other than those in Z. An action a_i is said to be i-*guarded* in P if each occurrence appears in the i-th argument S_i of some sub-expression $z(S_1, S_2, \ldots, S_n) \trianglelefteq P$.

Definition 1. *An expression P is called an n-*pearl *in* shell variables *Z if*

(**P1**) *P is Z-pure (and all $z \in Z$ have rank n).*
(**P2**) *In every sub-expression $z(S_1, S_2, \ldots, S_{n-1}, U) \trianglelefteq P$, for $z \in Z$, U has a free process variable, and all S_i are process constants such that $S_i \approx a_i.\mathsf{0}$.*
(**P3**) *There is at least one occurrence of some $z \in Z$ in P, and each action prefix a_i in P, for $i \geq 1$, is i-guarded.*

*An expression S is an n-*shell *if $P \trianglelefteq S$ for some n-pearl P and $\xi_n^Z(S) \approxeq \boldsymbol{A}_n$. A process p is an n-*noose *if there exists an n-shell S such that $p \equiv \xi_n^z(S)$.*

Since the size information n can be derived from the shell variables Z we will simply talk about *pearls* and *shells* in Z. Note that (P3) implies that a pearl can only contain observable actions a_i for $i \leq n$. Also, $\xi_n^Z(S) \approxeq \mathbf{A}_n$ means that shells S can at most have free variables in Z.

By definition, every n-noose p satisfies $p \approxeq \mathbf{A}_n$. The converse does not hold. Since nooses mix semantic and syntactic properties, they are not in general preserved by observational congruence. For instance, $E_2^2[a_0.\mathsf{0}, a_1.\mathsf{0}]$ is a 2-noose with shell (and pearl) $S \equiv \mu x_0.\,(a_0.\mathsf{0} + \tau.\,z_1(a_1.\mathsf{0}, x_0))$ and shell variables $Z =_{\mathrm{df}} \{z_1\}$, while $\mathbf{A}_2 \equiv \tau.\,(a_0.\mathsf{0} + a_1.\mathsf{0})$ which is observationally congruent to $E_2^2[a_0.\mathsf{0}, a_1.\mathsf{0}]$ is not a noose. In general, every $E_n^n[\tilde{a}] \approxeq \mathbf{A}_n$ is an n-noose but $\mathbf{A}_n$ is not. Our incompleteness result is based on the observation that, although $E_n^n[\tilde{a}]$ may be transformed under equational reasoning, the property of being an n-noose is hard to break up. Once infected by n-nooses with large n, equational transformations in rank-restricted Horn theories cannot get rid of them. The reason for this is that such proofs always factorise through shells which must be preserved by observational congruence. This is the content of the following propositions.

Proposition 3. *Let E and F be two schemes such that $\theta(E) \approxeq \theta(F)$ for all instantiations θ. Then, E is an n-shell iff F is an n-shell.*

Proposition 4. *Let E be a scheme of maximal recursion depth rd in which all free variables have maximal rank rk. Suppose $rd < 2$ and $rk < n - 2$, or $rd < n - 1$ and $rk < 2$. Then, every instantiation θ such that $\theta(E)$ is an n-noose can be factorised as $\theta = \xi_n^Z \circ \theta'$ for some instantiation θ' and rank n variables Z in such a way that $\theta'(E)$ is a shell in shell variables Z.*

The proofs of Props. 3 and 4 involve various auxiliary results about the properties of pearls under semantic equivalence transformations and decomposition by second-order unification. Based on Props. 3 and 4, the non-axiomatisability result is easily argued using an *intensional* equivalence $p \approxeq_n q$, defined by the condition that $p \approxeq q$ and either both p and q have an n-noose or none of them has. First, one observes that any rank-restricted equational axiomatisation that is sound for $\approxeq$ must also be sound in the intensional sense for large enough n.

Theorem 2. *Let $\mathcal{A}$ be a finite second-order equational axiomatisation of maximal variable rank 1 which is sound for $\approxeq$. Then, there exists a natural number m such that for all $n \geq m$, $\mathcal{A} \vdash p = q$ implies $p \approxeq_n q$.*

Proof. Choose m to be larger than the maximal nesting depth of recursions (or the maximal number of free variables) occurring in sub-expressions of any equation of $\mathcal{A}$. Since $\approxeq_n$ is an equivalence, the rules of reflexivity, transitivity and symmetry are sound for $\approxeq_n$. Hence, the statement of Thm. 2 follows directly by induction on the length of derivations $\mathcal{A} \vdash p = q$ if we can show that all equational axioms are sound for $\approxeq_n$. To this end, suppose $E = F$ is an axiom of $\mathcal{A}$ and $p \equiv \theta(E)$ and $q \equiv \theta(F)$ for some instantiation θ. If p is an n-noose, then, by Prop. 4 and $n \geq m$, there exists an instantiation θ' such that $\theta'(E)$ is an n-shell and $\theta = \xi_n \circ \theta'$. Since $E = F$ is sound for $\approxeq$, the expression $\theta'(F)$ is observationally congruent to the n-shell $\theta'(E)$. This implies $\theta'(F)$ is an n-shell by Prop. 3, whence $\theta(F) \equiv \xi_n(\theta'(F)) \equiv q$ is an n-noose. This proves $p \approxeq_n q$. $\square$

Now, for any natural number n, we have $E_n^n[\tilde{a}] \approxeq \mathbf{A}_n$. Since $E_n^n[\tilde{a}]$ is an n-noose but $\mathbf{A}_n$ is not, by Thm. 2, the sound equation $E_n^n[\tilde{a}] = \mathbf{A}_n$, for large enough n, is not derivable in any finite rank 1 equational axiom system. In other words, any finite system of second-order equational axioms of maximal variable rank 1 which is sound for $\approxeq$ is incomplete. This corollary to Thm. 2 is in itself not surprising since it is already known, e.g., from the work of Sewell [26], that pure equational logic is not sufficient to finitely axiomatise (strong) bisimulation on μ-expressions finitely. On the other hand, it is an open problem whether $\approxeq$ can be axiomatised in the more powerful setting of equational Horn logic. As we have seen in Section 3, this is indeed possible for the guarded fragment.

In the following we use Thm. 2 to derive two negative results, showing that the two well-known Horn-rules considered by Milner [22] and Bloom/Ésik [6] are incomplete. This is because these rules maintain the intensional equivalence $\approxeq_n$.

Milner's Rule R2. Consider Milner's only rule, the folding rule R2, whose soundness depends on a guardedness side condition:

$$\mathsf{R2} \quad \frac{y = z(y)}{y = \mu x.\, z(x)} \; z \text{ guarded}$$

Theorem 3. *There is no finite rank* 1 *equational extension of Milner's R2 rule (including Milner's axiomatisation without C4) which is sound and complete for* $\approxeq$ *on unguarded processes.*

Proof. We may assume that any application of R2 instantiates schematic variable z with a nontrivial guarded context, i.e., in which its argument is indeed called behind observable actions. For any instantiation of R2 in which $\theta(z)$ does not invoke its argument (i.e., does not use call-back \$1) we have $\theta(z)[p] \equiv \theta(z)$, for any p. One can thus derive $\mu x.\, \theta(z)[x] = \theta(z)$ via the unfolding rule R1, and from there obtain the conclusion $\theta(y) = \mu x.\, \theta(z)[x]$ via standard equational reasoning from the premise $\theta(y) = \theta(z)[\theta(y)]$. In other words, rule R2 becomes redundant for trivial instantiations. Assuming $\theta(z)$ contains \$1 we show that R2 can never produce in its conclusion a term that is an n-noose, for any $n \geq 1$.

Suppose R2 is used with instantiation θ such that $\theta(y)$ is an n-noose. By soundness, we would have $\mu x.\, \theta(z)[x] \approxeq \theta(y) \approxeq \theta(z)[\theta(y)]$. However, this cannot be true. The argument $\theta(y)$ of $\theta(z)$ is guarded by an observable action, say b, so that the right-hand side $\theta(z)[\theta(y)]$ has all noose actions a_i from the argument $\theta(y)$ accessible behind b. However, the process $\theta(y)$ on the left-hand side, being an n-noose and thus observationally congruent to $\mathbf{A}_n$, does not perform two actions in sequence. Thus, rule R2 is never applicable when y is instantiated with a process that is an n-noose.

Now suppose that R2 is instantiated so that $\theta(\mu x.\, z(x)) \equiv \mu x.\, \theta(z)[x]$ is an n-noose. Since $\theta(z)$ must have a guarded call-back, this recursion would be able to perform an infinite sequence of actions, which is not possible for nooses. Hence, R2 is sound for $\approxeq_n$, from which Thm. 2 obtains Thm. 3. □

Bloom and Ésik's "GA-implication" Rule. Bloom and Ésik [6] presented an implicational axiomatisation using the single rule scheme

$$\mathsf{GA} \quad \frac{\mu x.\, w_1(x,x) = \mu x.\, w_2(x,x)}{\mu x.\, w_1(x,x) = \mu x.\, w_2(x, \mu y.\, w_1(x,y))}$$

in two rank 2 variables w_1 and w_2, which is sound and complete for strong bisimulation. As reported in [26], this theory, together with the usual τ-laws [22], is also complete for $\approxeq$. Bloom/Ésik's system is a pure equational theory without side-conditions. However, it still depends on the infinitary congruence rule C4. Again, the reason is that GA preserves large nooses.

Proposition 5. *If Bloom/Ésik's rule GA is sound for $\approxeq$, then it is also sound for $\approxeq_n$, for all $n \geq 5$.*

By Thm. 2, every sound finite rank 1 equational axiom system is sound for $\approxeq_n$, for some large enough n. Since, by Prop. 5, rule GA still preserves $\approxeq_n$, no finite sound equational extension of GA in rank 1 can derive the equality $E_n^n[\tilde{a}] = \mathbf{A}_n$ which is not sound under $\approxeq_n$. Thus, the following theorem holds:

Theorem 4. *There is no finite rank* 1 *equational extension of Bloom/Ésik's rule GA which is sound and complete for $\approxeq$ on unguarded processes.*

5 Discussion, Conclusions and Future Work

We studied the logical basis for equational reasoning about observational congruence on μ-expressions. Pure equational logic is too inexpressive for bisimulation semantics on μ-expressions, while second-order equational Horn logic with its generic rule schemes seems powerful enough to admit finite axiomatisations. Indeed, it is well known that finite axiomatisations for this purpose must employ (second-order) rule schemes [26]. However, this does not mean that those axiomatisations are necessarily pure Horn systems. Specifically, as pointed out here, the congruence rule C4 for the μ-binder is beyond Horn logic. Rule C4 is mostly implicit and taken for granted; however, it breaks the purity of Horn logic and the straightforward applicability of Prolog-style resolution techniques. In particular, it makes the convenient identification of object-level process variables and meta-level schematic variables ("shallow embedding") impossible. Under the traditional point of view, perhaps, the formal complications due to C4 may be considered minimal. Still, the question must be asked whether bisimulation-style equivalences on μ-expressions can in fact be axiomatised in pure Horn logic and thus enjoy the pleasant model-theoretic and proof-theoretic properties of this rather natural logical setting.

This paper undertook some important steps in answering this question. On the positive side, we showed that observational congruence $\approxeq$ can in fact be axiomatised in pure Horn logic for the fragment of *guarded* processes. On the negative side, we proved that Milner's rule R2 and Bloom/Ésik's rule GA, which are known to be complete in the presence of congruence rule C4, cannot be finitely extended by rank 1 equational axioms for *unguarded* processes to yield a complete system for $\approxeq$ without C4. The proof turned out to be highly technical and involved subtle issues in managing second-order unification. However, the effort is well spent since negative results in the more powerful setting of equational Horn logic are potentially more interesting than negative results for pure equational logic. We should mention that we believe that our results do not depend on the cardinality of the action set: provided there exists one action, we can replace the distinct actions a_i by pairwise non-congruent processes.

Our results suggest that pure equational Horn systems are intrinsically limited in dealing with unguarded processes, which applies to both strong bisimulation and observational congruence. For the latter, however, this is more serious since unguardedness across unobservable actions is nontrivial when these are generated dynamically from communication (as in CCS [21]) or hiding (as in CSP [17]). In fact, our work was triggered by failed attempts to obtain a complete axiomatisation of observational congruence for regular processes in the timed process algebras PMC [4] and CSA [10]. In those languages, unguarded processes carry nontrivial semantic behaviour and thus cannot be ignored in the axiomatisation. If it turned out that unguardedness cannot be Horn axiomatised, this would exhibit the intrinsically more difficult proof-theoretic nature of deterministically timed process algebras under observational abstraction.

We believe that the notions of pearls and nooses introduced in this paper can be extended to obtain a negative result for arbitrary rank k equational schemes. As currently defined, our E^n_k contexts would not survive the so-called "diagonal" or "double iteration" identity $\mu x.\, \mu y.\, w(x,y) = \mu x.\, w(x,x)$ (see, e.g., [12]), which has rank 2. Using this scheme together with rank 1 axioms $\mu x.(y + z(x)) = y + \mu x.\, z(x+y)$ and $\mu x.\, \tau.\, z(x) = \tau.\, \mu x.z(\tau.\, x)$ – as well as a finite list of other rank 1 equations for reasoning about processes with a single recursion – would be strong enough to prove $E^n_n[\tilde{a}] = \mathbf{A}_n$. However, we conjecture that by re-defining the E^n_k to rank $n{-}k$ so that $E^{i+1}_{j+1} =_{\mathrm{df}} \mu x_{i-j}.(a_{i-j}.\, x_{i-j} + \tau.\, E^{i+1}_j[\mathsf{\$1}, \mathsf{\$2}, \ldots, \mathsf{\$(i\text{-}j)}, x_{i-j}])$, for $j \geq 0$, and E^{i+2}_0 as before in Sec. 4, $E^n_n[\tilde{a}] = \mathbf{A}_n$ cannot be proved using rank 2 equations. We leave the general rank k result and the more difficult case of equational Horn rules other than $\mathsf{R2}$ and GA to future work. Future work shall also investigate whether categorical languages, such as the one used by Bloom and Ésik in [7], can help to simplify our technical framework.

References

1. Aceto, L., Chen, T., Fokkink, W., Ingolfsdottir, A.: On the axiomatizability of priority. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 480–491. Springer, Heidelberg (2006)
2. Aceto, L., Fokkink, W., Ingolfsdottir, A., Luttik, B.: Finite equational bases in process algebra: Results and open questions. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity. LNCS, vol. 3838, pp. 338–367. Springer, Heidelberg (2005)
3. Aceto, L., Jeffrey, A.: A complete axiomatization of timed bisimulation for a class of timed regular behaviours. TCS 152(2), 251–268 (1995)
4. Andersen, H.R., Mendler, M.: An asynchronous process algebra with multiple clocks. In: Sannella, D. (ed.) ESOP 1994. LNCS, vol. 788, pp. 58–73. Springer, Heidelberg (1994)
5. Baeten, J.C.M., Middelburg, C.A.: Process Algebra with Timing. Springer, Heidelberg (1998)
6. Bloom, S.L., Ésik, Z.: Iteration algebras. Foundations of Computer Science 3(3), 245–302 (1992)
7. Bloom, S.L., Ésik, Z.: Iteration Theories: The Equational Logic of Iterative Processes. EATCS Monographs in TCS. Springer, Heidelberg (1993)

8. Bloom, S.L., Ésik, Z.: Iteration algebras are not finitely axiomatizable. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 367–376. Springer, Heidelberg (2000)
9. Chen, T., Fokkink, W.: On finite alphabets and infinite bases III: Simulation. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 421–434. Springer, Heidelberg (2006)
10. Cleaveland, R., Lüttgen, G., Mendler, M.: An algebraic theory of multiple clocks. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 166–180. Springer, Heidelberg (1997)
11. Conway, J.H.: Regular Algebra and Finite Machines. Chapman & Hall, Australia (1971)
12. Ésik, Z.: The equational theory of fixed points with applications to generalized language theory. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 21–36. Springer, Heidelberg (2002)
13. Fokkink, W.: A complete equational axiomatization for prefix iteration. Information Processing Letters 52(6), 333–337 (1994)
14. Fokkink, W., Zantema, H.: Basic process algebra with iteration: Completeness of its equational axioms. The Computer J. 37(4), 259–267 (1994)
15. van Glabbeek, R.: The linear time–branching time spectrum. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 278–297. Springer, Heidelberg (1990)
16. van Glabbeek, R.: A complete axiomatization for branching bisimulation congruence of finite state behaviours. In: Borzyszkowski, A.M., Sokolowski, S. (eds.) MFCS 1993. LNCS, vol. 711, pp. 473–484. Springer, Heidelberg (1993)
17. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs (1985)
18. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Inform. & Comp. 110(2), 366–390 (1994)
19. Mendler, M., Lüttgen, G.: Is observational congruence on μ-expressions axiomatisable in equational Horn logic? Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik, Techn. Rep. No. 72, Univ. of Bamberg (June 2007)
20. Milner, R.: A complete inference system for a class of regular behaviours. J. of Computer and System Sciences 28(3), 439–466 (1984)
21. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
22. Milner, R.: A complete axiomatisation for observational congruence of finite-state behaviours. Inform. & Comp. 81(2), 227–247 (1989)
23. Moller, F.: Axioms for Concurrency. PhD thesis, LFCS, Univ. of Edinburgh (1989), Also published as ECS-LFCS-89-84.
24. Moller, F.: The nonexistence of finite axiomatisations for CCS congruences. In: LICS'90, pp. 142–153. IEEE Computer Society Press, Los Alamitos (1990)
25. Nadathur, G., Miller, D.: Higher-order Horn clauses. JACM 37(4), 777–814 (1990)
26. Sewell, P.: Nonaxiomatisability of equivalences over finite state processes. Annals of Pure and Applied Logic 90, 163–191 (1997)

The *Must* Preorder Revisited
An Algebraic Theory for Web Services Contracts

Cosimo Laneve[1] and Luca Padovani[2]

[1] Department of Computer Science, University of Bologna
[2] Information Science and Technology Institute, University of Urbino

Abstract. We define a language for Web services contracts as a parallel-free fragment of CCS and we study a natural notion of compliance between clients and services in terms of their corresponding contracts. The induced contract preorder turns out to be valuable in searching and querying registries of Web services, it shows interesting connections with the must preorder, and it exhibits good precongruence properties when choreographies of Web services are considered. Our contract language may be used as a foundation of Web services technologies, such as WSDL and WSCL.

1 Introduction

Web services contracts are coarse-grained abstract descriptions to be used in workflows and business processes implemented in composite applications or portals. These descriptions are intended to be replaced by fine-grained implementations that provide complex Web services.

Current technologies for describing contracts specify the format of the exchanged messages – the *schema* –, the locations where the interactions are going to occur – the *interface* –, the transfer mechanism to be used (i.e. SOAP-RPC, or others), and the pattern of conversation of the service – the *behavior*. For example, the Web Service Description Language (WSDL) [11,10,9] defines simple behaviors: one-way (asynchronous) and request/response (synchronous) ones. The Web Service Conversation Language (WSCL) [2] extends WSDL behaviors by allowing the description of arbitrary, possibly cyclic sequences of exchanged messages between communicating parties. (Other languages, such as the Abstract business processes in the Web Service Business Execution Language (WS-BPEL) [1], provide even more detailed descriptions because they also define the subprocess structure, fault handlers, etc. We think that such descriptions are much too concrete to be used as contracts.)

Documents describing WSDL and WSCL contracts can be published in registries [3,12] so that Web services can be *searched* and *queried*. These two basic operations assume the existence of some notion of contract equivalence. The lack of a formal characterization of contracts only permits excessively demanding notions of equivalence such as syntactical equality. In fact, it makes perfect sense to further relax the equivalence into a *subcontract preorder* (denoted by $\preceq$ in this paper), so that Web services exposing "larger" contracts can be *safely* returned as results of queries for Web services with "smaller" contracts. The purpose of this paper is to define precisely what "larger" and "smaller" mean, as well as to define which safety property we wish to preserve when substituting a service exposing a contract with a service exposing a larger contract.

In our formalization, contracts are pairs $I : \sigma$, where I is the *interface*, i.e. the set of names used by the service for responses and requests, and σ is the *behavior*, i.e. the conversation pattern (it is intended that the names occurring in σ are included into I). Our investigation abstracts away from the syntactical details of schemas as well as from those aspects that are too oriented to the actual implementations, such as the definition of transmission protocols; all of such aspects may be easily integrated on top of the formalism. We do not commit to a particular interpretation of names either: they can represent different typed channels on which interaction occurs or different types of messages. Behaviors are sequences of request or response actions at specific names, possibly combined by means of two choice operators. The *external choice* "+" means that it is the interacting part that decides which one of alternative behaviors to carry on; the *internal choice* "$\oplus$" means that the it is the part exposing the contract that decides how to proceed. Recursive behaviors are also admitted. As a matter of facts, contracts are CCS (without τ's) processes that do not manifest internal moves and the parallel structure [17,19].

To equip our contracts with a subcontract preorder $\preceq$, we commit to a testing approach. We define client satisfaction as the ability of the client to complete successfully the interaction with the service; "successfully" meaning that the client never gets stuck (this notion is purposefully asymmetric as client's satisfaction is our main concern). The preorder arises by comparing the sets of clients satisfied by services. The properties enjoyed by the $\preceq$ preorder are particularly relevant in the context of Web services. For example, a service exposing the contract $\{a,b,c\} : \overline{a}.c \oplus \overline{b}.c$ is one that sends a message on a or on b – the choice is left to the service – and then waits for a response on c. A client compatible with such service must be prepared to accept messages from both a and b. Hence, such a client will also be compatible with services exposing either $\{a,b,c\} : \overline{a}.c$ or $\{a,b,c\} : \overline{b}.c$, which behave more deterministically than the original service, or even $\{a,b,c\} : \overline{a}.c + \overline{b}.c$, which leaves the choice to the client. This is expressed as $\{a,b,c\} : \overline{a}.c \oplus \overline{b}.c \preceq \{a,b,c\} : \overline{a}.c$ and $\{a,b,c\} : \overline{a}.c \oplus \overline{b}.c \preceq \{a,b,c\} : \overline{a}.c + \overline{b}.c$. Notice that $\{a,b,c\} : \overline{a}.c \preceq \{a,b,c\} : \overline{a}.c + \overline{b}.c$ should not hold. For example a client with contract $\{a,b,c\} : a.\overline{c} + b.a.\overline{c}$ successfully completes when interacting with $\{a,b,c\} : \overline{a}.c$, but it may fail when interacting with $\{a,b,c\} : \overline{a}.c + \overline{b}.c$. However, a client interacting with $\{a,c\} : \overline{a}.c$, that is never interacting on b, cannot fail with $\{a,b,c\} : \overline{a}.c + \overline{b}.c$. Namely, if a service is extended *in width* with *new* functionalities, the old clients will still complete successfully with the new service. Similarly, extensions *in depth* of a service should allow old clients to complete: $\{a,c\} : \overline{a}.c \preceq \{a,b,c\} : \overline{a}.c.\overline{b}.c$. To the best of our knowledge, there is no process semantics corresponding to $\preceq$ in the literature. However, the restriction of $\preceq$ to contracts with the same interface is a well-known semantics: the *must-testing preorder* [18,19,15].

Our investigation about a semantics for contracts also addresses the problem of determining, given a client exposing a certain behavior, the smallest (according to $\preceq$) service contract that satisfies the client – the *principal dual contract*. This contract, acting like a *principal type* in type systems, guarantees that a query to a Web services registry is answered with the largest possible set of compatible services in the registry's databases. Technically, computing the dual contract is not trivial because it cannot be reduced to a swapping of requests and responses, and external and internal choices. For example, applying this swapping to the contracts $\{a, b, c\} : a.\overline{b}+a.\overline{c}$ and $\{a, b, c\} : a.(\overline{b}\oplus\overline{c})$, which happen to be equivalent according to $\preceq$, would produce the contracts $\{a, b, c\} : \overline{a}.b \oplus \overline{a}.c$ and $\{a, b, c\} : \overline{a}.(b + c)$, respectively, but only the latter one does actually satisfy the clients exposing one of the two original contracts. As another example, the dual contract of $\{a\} : a$ cannot simply be $\{a\} : \overline{a}$, because a service with contract $\{a\} : \overline{a} \oplus (\overline{a} + a)$ satisfies the original contract and yet $\{a\} : \overline{a} \not\preceq \{a\} : \overline{a} \oplus (\overline{a} + a)$.

We also study the application of our theory of contracts to choreographies of Web services [16]. A choreography is an abstract specification of several services that run in parallel and communicate with each other by means of private names. (Actually, the behavior of each service may be synthesized out of a global description [5,4]). We show that $\preceq$ is robust enough so that, replacing an abstract specification with an implementation that is related to the specification by means of $\preceq$, the observable features of the choreography as a whole are preserved.

Related work. This research was inspired by "CCS without τ's" [19] and by Hennessy's model of acceptance trees [14,15]. Our contracts are an alternative representation of acceptance trees. The use of formal models to describe communication protocols is not new (see for instance the exchange patterns in SSDL [21], which are based on CSP and the π-calculus), nor is it the use or CCS processes as behavioral types (see [20] and [8]). However, to the best of our knowledge the subcontract relation $\preceq$ is original. Although it resembles the must preorder (and it reduces to the must preorder when the interfaces are large enough), $\preceq$ arises from a notion of compliance that significantly differs from the notion of "passing a test" in the testing framework [18] and that more realistically describes well-behaved clients of Web services. The *width* extension property enjoyed by $\preceq$ is closely related to subtyping in object-oriented programming languages. The works that are more closely related to ours are by Carpineti *et al.* [6], by Castagna *et al.* [7] and the ones on *session types*, especially [13] by Gay and Hole. In [6] the subcontract relation (over finite contracts) exhibits all of the desirable properties illustrated in the introduction. Unfortunately, such relation turns out to be non transitive as the preorder arises as a syntactic notion. Transitivity, while not being strictly necessary as far as querying and searching are concerned (in fact, the coinductive notion of compliance defined in [6] would suffice) has two main advantages: on the practical side it allows databases of Web services contracts to be organized in accordance with the subcontract relation, so as to reduce the run time spent for executing queries. The transitive closure of a query can be precomputed when a new service is registered. On the theoretical side, it is a fundamental prerequisite when contracts are considered as (behavioral) types for typing processes: the subcontract relation is just the subsumption rule. The transitivity problem has been also addressed in [7] in a more general way. However, the authors of [7] must introduce a rather powerful construct (the *filter*) which prevents potentially dangerous interactions. Roughly speaking, a filter actively mediates the client/service interaction at run time, by dynamically changing the interface of the service as it is seen from the client. With respect to [13] our contract language is much simpler and it can express more general forms of interaction. While the language defined in [13] supports first-class sessions and name passing, it is purposefully tailored so that the transitivity problems mentioned above are directly avoided at the language level. This restricts the subcontract relation in such a way that internal and external choices can never be related (hence, $\{a, b\} : a \oplus b \preceq \{a, b\} : a + b$ does *not* hold).

Structure of the paper. In Section 2 we formally define our language for contracts, the corresponding transition relation and the compliance of a client with a service. In Section 3 we study the subcontract preorder and its relationship with the must preorder. In Section 4 we analyze the problem of determining the principal dual contract. In Section 5 we discuss the application of the subcontract preorder to choreographies. Section 6 discusses the expressivity of our contracts by showing the encoding of a WSCL conversation into our language, it draws our conclusions, and hints at future work.

2 The Contract Language

The syntax of contracts uses an infinite set of *names* $\mathcal{N}$ ranged over by a, b, c, ..., and a disjoint set of *co-names* $\overline{\mathcal{N}}$ ranged over by $\overline{a}$, $\overline{b}$, $\overline{c}$, ... We use the term *action* for referring to names and co-names without distinction. We let $\overline{\overline{a}} = a$, we use $\alpha, \beta, \ldots$ to range over $\mathcal{N} \cup \overline{\mathcal{N}}$, and we use $\varphi, \psi, \ldots$ to range over $(\mathcal{N} \cup \overline{\mathcal{N}})^*$. *Contracts* are pairs $I : \sigma$ where I, called *interface*, is a finite subset of $\mathcal{N}$ representing the set of names on which interaction occurs, whereas σ, called *behavior*, is defined by the following grammar:

$$\begin{array}{lll}
\sigma ::= & & \\
\quad | & \mathbf{0} & \text{(null)} \\
\quad | & \alpha.\sigma & \text{(prefix)} \\
\quad | & x & \text{(variable)} \\
\quad | & \sigma + \sigma & \text{(external choice)} \\
\quad | & \sigma \oplus \sigma & \text{(internal choice)} \\
\quad | & \mathsf{rec}\ x.\sigma & \text{(recursion)}
\end{array}$$

The behavior $\mathbf{0}$ defines the empty conversation; the behavior $a.\sigma$ defines a conversation protocol whose initial activity is to accept a request on a and continuing as σ; the behavior $\overline{a}.\sigma$ defines a conversation protocol whose initial activity is to send a response to a and continuing as σ. Behaviors $\sigma + \sigma'$ and $\sigma \oplus \sigma'$ define conversation protocols that follow either the conversation σ or σ'; in $\sigma + \sigma'$ the choice is left to the remote party, in $\sigma \oplus \sigma'$ the choice is made locally. For example, $\mathtt{Login}.(\overline{\mathtt{Continue}} + \overline{\mathtt{End}})$ describes the conversation protocol of a service that is ready to accept `Login`s and will `Continue` or `End` the conversation according to client's request. This contract is different from $\mathtt{Login}.(\overline{\mathtt{Continue}} \oplus \overline{\mathtt{End}})$ where the decision whether to continue or to end is taken by the service. The behavior $\mathsf{rec}\ x.\sigma$ defines a possibly recursive conversation protocol whose recurrent pattern is σ. A (free) occurrence of the variable x in σ stands for the whole $\mathsf{rec}\ x.\sigma$. In the following we write $\mathbf{\Omega}$ for $\mathsf{rec}\ x.x$.

Let $\mathtt{names}(\sigma)$ be the set of names a such that either a or $\overline{a}$ occur in σ. We always assume that $\mathtt{names}(\sigma) \subseteq I$ holds for every $I : \sigma$. With an abuse of notation we write $\mathtt{names}(\varphi)$ for the set of names occurring in φ.

Behaviors retain a *transition relation* that is inductively defined by the rules

$$\alpha.\sigma \xrightarrow{\alpha} \sigma \qquad \sigma_1 \oplus \sigma_2 \longrightarrow \sigma_1 \qquad \frac{\sigma_1 \xrightarrow{\alpha} \sigma_1'}{\sigma_1 + \sigma_2 \xrightarrow{\alpha} \sigma_1'} \qquad \frac{\sigma_1 \longrightarrow \sigma_1'}{\sigma_1 + \sigma_2 \longrightarrow \sigma_1' + \sigma_2}$$

$$\mathsf{rec}\ x.\sigma \longrightarrow \sigma\{^{\mathsf{rec}\ x.\sigma}/_{x}\}$$

plus the symmetric rules for $\oplus$ and $+$. This operational semantics is exactly the same as CCS without τ's [19]. In particular, the rules for $\oplus$ say that the behavior $\sigma_1 \oplus \sigma_2$ may exhibit σ_1 or σ_2 through an internal, unlabeled transition. The behavior $\sigma_1 + \sigma_2$ may exhibit σ_1 or σ_2 only after performing a visible, labeled transition of σ_1 or σ_2, respectively; internal transitions do not modify the external choice. A recursive behavior $\mathsf{rec}\ x.\sigma$ unfolds to $\sigma\{^{\mathsf{rec}\ x.\sigma}/_{x}\}$ with an internal transition. We write $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$; $\sigma \overset{\alpha}{\Longrightarrow} \sigma'$ for $\sigma \Longrightarrow\xrightarrow{\alpha}\Longrightarrow \sigma'$; $\sigma \overset{\alpha_1 \cdots \alpha_n}{\Longrightarrow} \sigma'$ if $\sigma \overset{\alpha_1}{\Longrightarrow} \cdots \overset{\alpha_n}{\Longrightarrow} \sigma'$; $\sigma \overset{\varphi}{\Longrightarrow}$ if there exists σ' such that $\sigma \overset{\varphi}{\Longrightarrow} \sigma'$. We write $\sigma\uparrow$ if σ has an infinite internal computation $\sigma = \sigma_0 \longrightarrow \sigma_1 \longrightarrow \sigma_2 \longrightarrow \cdots$ and $\sigma\downarrow$ if not $\sigma\uparrow$. We write $\sigma \downarrow \varphi$ if $\sigma \overset{\varphi}{\Longrightarrow} \sigma'$ implies $\sigma'\downarrow$. Finally, we write $\sigma \uparrow \varphi$ if $\sigma \overset{\varphi}{\Longrightarrow} \sigma'$ and $\sigma'\uparrow$ for some σ'. For example $\mathbf{\Omega}\uparrow$, $\mathsf{rec}\ x.a + x\uparrow$, and $\mathsf{rec}\ x.(a.x + b.x) \downarrow \varphi$ for every $\varphi \in \{a, b\}^*$. Let $\mathtt{init}(\sigma) \overset{\text{def}}{=} \{\alpha \mid \sigma \overset{\alpha}{\Longrightarrow}\}$.

A basic use of contracts is to verify whether a client protocol is consistent with a service protocol. This consistency, called behavioral compliance in the following, requires two preliminary definitions: that of communicating behaviors and that of matching:

– The notion of *communicating behaviors* extends the transition relation $\longrightarrow$ to pairs of behaviors as follows:

$$\frac{\rho \longrightarrow \rho'}{\rho \mid \sigma \longrightarrow \rho' \mid \sigma} \qquad \frac{\sigma \longrightarrow \sigma'}{\rho \mid \sigma \longrightarrow \rho \mid \sigma'} \qquad \frac{\rho \xrightarrow{\alpha} \rho' \quad \sigma \xrightarrow{\overline{\alpha}} \sigma'}{\rho \mid \sigma \longrightarrow \rho' \mid \sigma'}$$

– The notion of *matching* is modeled using a special name e for denoting the successful termination of a party ("e" stands for `end`). By "success" we mean the ability to always reach a state in which e can be emitted.

Definition 1 (Behavioral compliance). *Let* $\mathsf{e} \notin \mathtt{names}(\sigma)$*. The (client) behavior* ρ *is* compliant with *the (service) behavior* σ*, written* $\rho \dashv \sigma$*, if* $\rho \mid \sigma \Longrightarrow \rho' \mid \sigma'$ *implies*

1. *if* $\rho' \mid \sigma' \not\longrightarrow$*, then* $\{\mathsf{e}\} \subseteq \mathtt{init}(\rho')$*;*
2. *if* $\sigma'\uparrow$*, then* $\{\mathsf{e}\} = \mathtt{init}(\rho')$*.*

According to the notion of behavioral compliance, if the client-service conversation terminates, then the client is in a successful state (it will emit an e-name). For example, $a.\mathsf{e} + b.\mathsf{e} \dashv \overline{a} \oplus \overline{b}$ and $a.\mathsf{e} \oplus b.\mathsf{e} \dashv \overline{a} + \overline{b}$ but $a.\mathsf{e} \oplus b.\mathsf{e} \not\dashv \overline{a} \oplus \overline{b}$ because of the computation $a.\mathsf{e} \oplus b.\mathsf{e} \mid \overline{a} \oplus \overline{b} \Longrightarrow a.\mathsf{e} \mid \overline{b} \not\longrightarrow$ where the client waits for an interaction on a in vain. Similarly, the client must reach a successful state if the conversation does not terminate but the divergence is due to the service. In this case, however, the compliance relation takes into account the subtleties that divergence implies. In particular, the client is constrained to autonomously reach a state where the only possible action is e. The practical justification of such a notion of compliance derives from the fact that connection-oriented communication protocols (like those used for interaction with Web services) typically provide for an explicit end-of-connection signal. Consider for example the client $\mathsf{e} + \overline{a}.\mathsf{e}$. Intuitively this client tries to send a request on the name a, but it can also succeed if the service rejects the request. So $\mathsf{e} + \overline{a}.\mathsf{e} \dashv \mathbf{0}$ because the client can detect the fact that the service is not ready to interact on a. The same client interacting with a diverging service would have no way to distinguish a service that is taking a long time to accept the request from a service that is perpetually performing internal computations, hence $\mathsf{e} + \overline{a}.\mathsf{e} \not\dashv \mathbf{\Omega}$. On the theoretical side, we will see that such a notion of compliance makes $\mathbf{\Omega}$ the "smallest service" (the one a client can make the least number of assumptions on), and this property will be fundamental in the definition of principal dual contract in Section 4.

Our notion of behavioral compliance enjoys the following "subject reduction" property, stating that given two compliant behaviors ρ and σ, every residual behaviors ρ' and σ' are also compliant.

Proposition 1. *If* $\rho \dashv \sigma$ *and* $\rho \mid \sigma \longrightarrow \rho' \mid \sigma'$*, then* $\rho' \dashv \sigma'$*.*

3 The Subcontract Relation

The behavioral compliance is actually a basic test for investigating services. Following De Nicola and Hennessy's approach to process semantics [18], this test induces a preorder on services on the basis of the set of clients that comply with a given service.

Definition 2 (Contract semantics). *Let* $[\![I : \sigma]\!] \stackrel{\text{def}}{=} \{J : \rho \mid J \subseteq I \text{ and } \rho \dashv \sigma\}$*. A contract* $I : \sigma$ *is a* subcontract *of* $I' : \sigma'$*, written* $I : \sigma \preceq I' : \sigma'$*, if and only if* $[\![I : \sigma]\!] \subseteq [\![I' : \sigma']\!]$*. Let* $\simeq$ *be* $\preceq \cap \succeq$*.*

For example, $[\![\emptyset : \mathbf{\Omega}]\!] = \{\emptyset : \mathsf{e}, \emptyset : \mathsf{rec}\ x.\mathsf{e} + x, \dots\}$ and $[\![\{a, b\} : a \oplus b]\!] = \{\emptyset : \mathsf{e}, \emptyset : \mathsf{rec}\ x.\mathsf{e} + x, \{a\} : \mathsf{e}, \{b\} : \mathsf{e}, \{a, b\} : \mathsf{e}, \{a, b\} : \overline{a}.\mathsf{e} + \overline{b}.\mathsf{e}, \dots\}$. It turns out that clients in $[\![\emptyset : \mathbf{\Omega}]\!]$ comply with every other service, hence the service $\emptyset : \mathbf{\Omega}$ is the smallest one. As usual it is easier to figure out inequalities: $\{a, b\} : a \not\preceq \{a, b\} : a.b$ because $\overline{a}.(\mathsf{e}+\overline{b}) \dashv a$ but $\overline{a}.(\mathsf{e}+\overline{b}) \not\dashv a.b$; $\{a, b\} : a \not\preceq \{a, b\} : a+b$ because $\mathsf{e}+\overline{b} \dashv a$ but $\mathsf{e} + \overline{b} \not\dashv a + b$.

The next proposition states the desirable properties listed in Section 1 in a formal way. Notice that for most of them it would be possible to weaken the premises, but we prefer this presentation as it highlights more clearly the relevant features of $\preceq$. The proofs are are relatively easy (see the alternative characterization of $\preceq$ in Definition 3).

Proposition 2. *Let $I : \sigma$ and $I' : \sigma'$ be contracts such that $I \cap I' = \emptyset$. Then:*

1. *if $J : \rho \preceq I \cup I' : \sigma$ and $J : \rho \preceq I \cup I' : \sigma'$, then $J : \rho \preceq I \cup I' : \sigma \oplus \sigma'$;*
2. *if $\sigma\downarrow$ and $\sigma'\downarrow$, then $I \cup I' : \sigma \preceq I \cup I' : \sigma + \sigma'$;*
3. *if $\sigma'\downarrow$, then $I \cup I' : \sigma\{^{\mathbf{0}}/_x\} \preceq I \cup I' : \sigma\{^{\sigma'}/_x\}$;*
4. *$I : \sigma \preceq I \cup I' : \sigma$.*

Item 1 states that $I \cup I' : \sigma \oplus \sigma'$ is the largest contract that satisfies the clients that are compliant with both $I \cup I' : \sigma$ and $I \cup I' : \sigma'$. Namely, $[\![I \cup I' : \sigma]\!] \cap [\![I \cup I' : \sigma']\!] = [\![I \cup I' : \sigma \oplus \sigma']\!]$. Item 2 gives sufficient conditions for width extensions of Web services: a Web service may be upgraded to offer additional functionalities without affecting the set of clients it satisfies, so long as such new functionalities regard names that were not present in the original service. In fact, it suffices to require $\mathtt{init}(\sigma') \cap I = \emptyset$ to establish the result. Contrary to item 1, $I \cup I' : \sigma + \sigma'$ is *not* the smallest contract that satisfies the clients that are compliant with either $I \cup I' : \sigma$ or $I \cup I' : \sigma'$. For example, $\{a, b\} : a.\mathsf{e} \oplus b.\mathsf{e} \in [\![\{a, b\} : \overline{a} + \overline{b}]\!]$ but $\{a, b\} : a.\mathsf{e} \oplus b.\mathsf{e} \notin [\![\{a, b\} : \overline{a}]\!]$ and $\{a, b\} : a.\mathsf{e} \oplus b.\mathsf{e} \notin [\![\{a, b\} : \overline{b}]\!]$, hence $[\![I \cup I' : \sigma]\!] \cup [\![I \cup I' : \sigma']\!] \subsetneq [\![I \cup I' : \sigma + \sigma']\!]$. Item 3 states a similar result, but for depth extensions, that is the ability to extend the conversation offered by a service, provided that the additional conversation occurs on names that were not present in the original service. The premises can be weakened as for item 2. In fact, item 2 can be seen as a special case of item 3, if we consider the contract $I \cup I' : \sigma + x$. Item 4 shows that merely increasing the names that a Web service can interact on does not affect the clients it satisfies.

As usual with testing semantics, it is hard to establish a relationship between two contracts because the sets $[\![I : \sigma]\!]$ are infinite. A direct definition of the preorder is therefore preferred.

Definition 3. *Let $\sigma \Downarrow \mathrm{R}$ if and only if $\sigma \Longrightarrow \sigma'$ and $\mathrm{R} = \mathtt{init}(\sigma')$. A* coinductive subcontract *is a relation $\mathcal{R}$ such that if $(I : \rho, J : \sigma) \in \mathcal{R}$, then $I \subseteq J$ and whenever $\rho\downarrow$ then*

1. *$\sigma\downarrow$, and*
2. *$\sigma \Downarrow \mathrm{R}$ implies $\rho \Downarrow \mathrm{R}'$ and $\mathrm{R}' \subseteq \mathrm{R}$, and*
3. *$\alpha \in I$ and $\sigma \overset{\alpha}{\Longrightarrow} \sigma'$ imply $\rho \overset{\alpha}{\Longrightarrow} \rho_1, \ldots, \rho \overset{\alpha}{\Longrightarrow} \rho_n$ for some $n \geq 1$ and $(I : \bigoplus_{1 \leq i \leq n} \rho_i, J : \sigma'_2) \in \mathcal{R}$.*

By this definition, a contract $I : \rho$ such that $\rho\uparrow$ is the smallest one with interface I. When $\rho\downarrow$, condition 1 constrains the larger contract $J : \sigma$ to converge as well, since clients might rely on the convergence of ρ to complete successfully. Condition 2 states that $J : \sigma$ must exhibit a more deterministic behavior: the lesser the number of ready sets is, the more deterministic the contract is. Furthermore, $J : \sigma$ should expose *at least* the same capabilities as the smaller one ($\textsc{r}' \subseteq \textsc{r}$). Condition 3 is perhaps the most subtle one, as it deals with all the possible derivatives of the smaller contract. The point is that $\{a, b, c\} : a.b + a.c \simeq \{a, b, c\} : a.(b \oplus c)$ since, after interacting on a, a client of the service on the left side of $\simeq$ is not aware of which state the service is in (it can be either b or c). Hence, we have to consider all of the possible derivatives after a, thus reducing to verifying $(\{a, b, c\} : (b + c) \oplus b \oplus c, \{a, b, c\} : b \oplus c) \in \mathcal{R}$ which trivially holds.

Theorem 1. *$\preceq$ is the largest coinductive subcontract relation.*

The proof is standard and heavily relies on the "unzipping" of a derivation $\rho \mid \sigma \overset{\varphi}{\Longrightarrow} \rho' \mid \sigma'$, which results into two sequences ψ and ψ' of actions such that $\rho \overset{\psi}{\Longrightarrow} \rho'$ and $\sigma \overset{\psi'}{\Longrightarrow} \sigma'$. Intuitively, ψ and ψ' contain (a subset of) the actions in φ interspersed with the actions on which (the derivatives of) ρ and σ have synchronized. When φ is empty, then $\overline{\psi} = \psi'$, where $\overline{\psi}$ is the co-sequence obtained from ψ by swapping names and co-names. By "zipping" we mean the inverse process whereby two or more derivations such as $\rho \overset{\psi}{\Longrightarrow} \rho'$ and $\sigma \overset{\psi'}{\Longrightarrow} \sigma'$ are combined to produce $\rho \mid \sigma \overset{\varphi}{\Longrightarrow} \rho' \mid \sigma'$. See [15] for a more detailed discussion.

We are not aware of any process semantics corresponding to $\preceq$ in the literature. However, if we restrict $\preceq$ to contracts with the same interface, we retrieve a well-known semantics: the *must-testing preorder* [15]. We recall the definition of the must preorder for the behaviors in Section 2.

Definition 4 (Must preorder [19]). *A sequence of transitions $\sigma_0 \mid \rho_0 \longrightarrow \sigma_1 \mid \rho_1 \longrightarrow \cdots$ is a* maximal computation *if either it is infinite or the last term $\sigma_n \mid \rho_n$ is such that $\sigma_n \mid \rho_n \not\longrightarrow$.*

Let $\mathtt{e} \notin \mathtt{names}(\sigma)$. Let σ $\mathtt{must}$ ρ if, for every maximal computation $\sigma \mid \rho = \sigma_0 \mid \rho_0 \longrightarrow \sigma_1 \mid \rho_1 \longrightarrow \cdots$, there exists $n \geq 0$ such that $\rho_n \xrightarrow{\mathtt{e}}$. We write $\sigma \sqsubseteq_{\mathtt{must}} \sigma'$ if and only if, for every ρ, σ $\mathtt{must}$ ρ implies σ' $\mathtt{must}$ ρ.

Before showing the precise relationship between $\preceq$ and $\sqsubseteq_{\mathtt{must}}$, let us comment on the differences between $\rho \dashv \sigma$ and σ $\mathtt{must}$ ρ. The $\mathtt{must}$ relation is such that σ $\mathtt{must}$ $\mathtt{e} + \rho$ holds for every σ, so that the observers of the form $\mathtt{e} + \rho$ are useless for discriminating between different (service) behaviors in $\sqsubseteq_{\mathtt{must}}$. However this is not the case for $\preceq$. For example $\mathtt{e} + a \not\dashv \overline{a}$ (whilst $\overline{a}$ $\mathtt{must}$ $\mathtt{e} + a$). In our setting it makes no sense to declare that $\mathtt{e} + a$ is compliant with $\overline{a}$ with the justification that, at some point in a computation starting from $\mathtt{e} + a \mid \overline{a}$, the client can emit $\mathtt{e}$. When a client and a service interact, actions cannot be undone. On the other hand we have $\mathtt{e} \oplus \mathtt{e} \dashv \boldsymbol{\Omega}$ and $\boldsymbol{\Omega}$ $\cancel{\mathtt{must}}$ $\mathtt{e} \oplus \mathtt{e}$. That is a (client) behavior compliant with a divergent (service) behavior is such that it is compliant with every (service) behavior, hence it is useless for discriminating between different (service) behaviors in $\preceq$. Historically, $\mathbf{\Omega}$ $\texttt{must}\!\!\!\!/\;\;$ $\mathsf{e} \oplus \mathsf{e}$ has been motivated by the fact that the divergent process may prevent the observer from performing the one internal reduction that leads to success. In a distributed setting this motivation is no longer sustainable, since client and service will usually run independently on different processors. Finally, consider a divergent (client) behavior ρ. In the $\texttt{must}$ relation such observer never succeeds unless $\rho \xrightarrow{\mathsf{e}}$. In the $\dashv$ relation such observer is compliant so long as all of its finite computations lead to a successful state. So, for example, the client behaviors $\mathsf{rec}\ x.a.\mathsf{e} + x$ and $a.\mathsf{e}$ have the same discriminating power as far as $\preceq$ is concerned.

Theorem 2. *Let $I = \texttt{names}(\sigma)$. $I : \sigma \preceq I : \tau$ if and only if $\sigma \sqsubseteq_{\texttt{must}} \tau$.*

4 Dual Contracts

We now turn our attention to the problem of querying a database of Web services contracts. The basic idea is that given a client $I : \rho$ we wish to find all the service contracts $J : \sigma$ such that $I \subseteq J$ and $\rho \dashv \sigma$. We can partly simplify the problem by computing *one particular service contract* $J_0 : \sigma_0$ such that $I \subseteq J_0$ and $\rho \dashv \sigma_0$ and then by taking all the services in the registry that are larger than this one. Actually, in order to maximize the number of service contracts returned as answer to the query, the *dual operator* of a (client) contract $I : \rho$ should compute a behavior $\rho^\perp$, so that $I : \rho^\perp$ is smallest service contract that satisfies the client contract $I : \rho$. We call such contract the *principal dual contract* of $I : \rho$.

It is convenient to restrict the definition of dual to those behaviors ρ that never lead to $\mathbf{0}$ without emitting e. For example, the behavior $a.\mathsf{e}+b.\mathbf{0}$ describes a client that succeeds if the service proposes $\overline{a}$, but that fails if the service proposes $\overline{b}$. As far as querying is concerned, such behavior is completely equivalent to $a.\mathsf{e}$. As another example, the degenerate client behavior $\mathbf{0}$ is such that no service will ever satisfy it. In general, if a client is unable to handle a particular action, like b in the first example, it should simply omit that action from its behavior. We say that a (client) contract $I : \rho$ is *canonical* if, whenever $\rho \overset{\varphi}{\Longrightarrow} \rho'$ is maximal, then $\varphi = \varphi'\mathsf{e}$ and $\mathsf{e} \notin \texttt{names}(\varphi')$. For example $\{a\} : a.\mathsf{e}$, $\{a\} : \mathsf{rec}\ x.a.x$, and $\emptyset : \mathbf{\Omega}$ are canonical; $\{a, b\} : a.\mathsf{e} + b.\mathbf{0}$ and $\{a\} : \mathsf{rec}\ x.a + x$ are not canonical.

To ease the definition of the dual we introduce a countable set of behavior names, called $\texttt{dual}(I, \rho)$, which are defined by equations $\texttt{dual}(I, \rho) \overset{\text{def}}{=} \sigma$, where I is finite and σ is a behavior containing behavior names $\texttt{dual}(I', \rho')$ instead of variables. In order for the definition to be well founded, we need to prove a preliminary finiteness result.

Proposition 3. *Continuations of prefixes in behaviors are finite. Namely, for every σ the set $\{\sigma' \mid$ there exist φ, α such that $\sigma \overset{\varphi}{\Longrightarrow}\xrightarrow{\alpha} \sigma'\}$ is finite.*

Notice that the set $\{\sigma' \mid \sigma \Longrightarrow \sigma'\}$ may be infinite. This is the case when $\sigma = \mathsf{rec}\ x.a + x$. However, the set $\{\sigma' \mid \mathsf{rec}\ x.a + x \Longrightarrow\xrightarrow{a} \sigma'\}$ is finite and it is $\{\mathbf{0}\}$. An immediate consequence of Proposition 3 is that behaviors are image finite. Namely, for every σ and α, the set $\{\sigma' \mid \sigma \Longrightarrow\xrightarrow{\alpha} \sigma'\}$ is finite.

Definition 5 (Dual contract). *Let $I : \rho$ be a canonical contract. Let* $\mathtt{co}(I) \stackrel{\text{def}}{=} \{\overline{a} \mid a \in I\}$. *The* dual *of ρ with respect to I, written* $\mathtt{dual}(I, \rho)$, *is defined as follows:*

$$\mathtt{dual}(I,\rho) \stackrel{\text{def}}{=} \begin{cases} \mathbf{\Omega}, & \textit{if } \mathtt{init}(\rho) = \{\mathtt{e}\} \\ \sum_{\rho \Downarrow \mathrm{R}, \mathrm{R}\setminus\{\mathtt{e}\}\neq\emptyset} \Big(\underbrace{\mathbf{0} \oplus}_{\textit{if } \mathtt{e} \in \mathrm{R}} \bigoplus_{\rho \Longrightarrow \xrightarrow{\alpha \in \mathrm{R}\setminus\{\mathtt{e}\}} \rho'} \overline{\alpha}.\mathtt{dual}(I, \rho') \Big) & \\ \underbrace{+ \Big(\mathbf{0} \oplus \bigoplus_{\alpha \in (I \cup \mathtt{co}(I)) \setminus \mathtt{init}(\rho)} \alpha.\mathbf{\Omega} \Big)}_{\textit{if } (I \cup \mathtt{co}(I)) \setminus \mathtt{init}(\rho) \neq \emptyset}, & \textit{otherwise} \end{cases}$$

The behavior $\mathtt{dual}(I, \rho)$ is well defined because the summands are always finite: the external summand is finite because behaviors manifest finitely many different ready sets; the internal summand is finite because of Proposition 3. It follows that from every equation $\mathtt{dual}(I,\rho) \stackrel{\text{def}}{=} \sigma$ it is possible to reach a finite set of behavior constants. It is folklore to transform such equations into recursive behaviors, thus conforming with the syntax of Section 2.

Few comments about $\mathtt{dual}(I, \rho)$, when $\mathtt{init}(\rho) \neq \{\mathtt{e}\}$, follow. In this case, the behavior ρ may autonomously transit to different states, each one offering a particular ready set. Thus the dual behavior leaves the choice to the client: this is the reason for the external choice in the second line. Once the state has been chosen, the client offers to the service a spectrum of possible actions: this is the reason for the internal choice in the first line (of the “otherwise” clause). The second line covers all the cases of actions that are allowed by the interface and that are not offered by the client. The point is that the dual operator must compute the principal (read, the smallest) service contract that satisfies the client, and the smallest convergent behavior with respect to a (finite) interface I is $\bigoplus_{\alpha \in I \cup \mathtt{co}(I)} \alpha$. The external choice in the second line distributes the proper dual contract over the internal choice of all the actions not allowed by the interface. The $\mathbf{0}$ summand accounts for the possibility that none of the actions not allowed by the interface is present. For example, $\mathtt{dual}(\{a\}, a.\mathtt{e}) = \overline{a}.\mathbf{\Omega} + (\mathbf{0} \oplus a.\mathbf{\Omega})$. The dual of a divergent (canonical) client is also well defined: $\mathtt{dual}(\{a\}, \mathsf{rec}\ x.a.\mathtt{e} + x) = \overline{a}.\mathbf{\Omega} + (\mathbf{0} \oplus a.\mathbf{\Omega})$. We notice that the definition also covers duals of nonterminating clients: $\mathtt{dual}(\{a\}, \mathsf{rec}\ x.a.x) = \overline{a}.\mathtt{dual}(\{a\}, \mathsf{rec}\ x.a.x) + (\mathbf{0} \oplus a.\mathbf{\Omega})$, namely $\mathtt{dual}(\{a\}, \mathsf{rec}\ x.a.x) \simeq \mathsf{rec}\ x.(\overline{a}.x + (\mathbf{0} \oplus a.\mathbf{\Omega}))$.

Theorem 3. *Let $I : \rho$ be a canonical contract. Then:*

1. $\rho \dashv \mathtt{dual}(I, \rho)$;
2. *if* $\rho \dashv \sigma$ *and* $\mathtt{names}(\sigma) \subseteq I$, *then* $I : \mathtt{dual}(I, \rho) \preceq I : \sigma$.

5 Choreographies

A *choreography* is meant to describe the parallel composition of n services (called participants) that communicate with each other by means of private names and with the external world by means of public names. Standard languages for describing choreographies, such as the Web Service Choreography Description Language (WS-CDL [16]), describe choreography activities, communications, and their mutual dependencies from a global perspective. From such descriptions it is possible to synthesize the so-called *end-point projections*, namely the behavioral specifications of the single participants, provided that the global description respects some fundamental constraints (see for example [5] and [4]).

In this work, we identify a choreography with the composition of its end-point projections, which are represented as contracts. Formally, a choreography is a term

$$\Sigma ::= (I_1 : \sigma_1 \mid \cdots \mid I_n : \sigma_n) \setminus L$$

where L is a finite subset of names representing the private names of the choreography. We write $\Sigma[i \mapsto J : \rho]$ for the choreography that is the same as Σ except that (the contract of) the i-th participant has been replaced by $J : \rho$. We also write Σ_L whenever we want to recall the private ports of the choreography.

The transition relation of choreographies is defined using that of behaviors by the following rules:

$$\frac{\sigma \xrightarrow{\alpha} \sigma' \qquad \mathtt{names}(\alpha) \notin L}{\Sigma_L[i \mapsto I : \sigma] \xrightarrow{\alpha} \Sigma_L[i \mapsto I : \sigma']} \qquad \frac{\sigma \longrightarrow \sigma'}{\Sigma_L[i \mapsto I : \sigma] \longrightarrow \Sigma_L[i \mapsto I : \sigma']}$$

$$\frac{i \neq j \quad \sigma \xrightarrow{\alpha} \sigma' \quad \rho \xrightarrow{\overline{\alpha}} \rho' \quad \mathtt{names}(\alpha) \in L}{\Sigma_L[i \mapsto I : \sigma][j \mapsto J : \rho] \longrightarrow \Sigma_L[i \mapsto I : \sigma'][j \mapsto J : \rho']}$$

Having provided choreographies with a transition relation, the notions of *convergence*, *divergence*, and *ready set* can be immediately extended to choreographies from their previous definitions. Similarly, the notion of behavioral compliance may be extended in order to relate the behavior of a client with (the behavior of) a choreography, which we denote by $\rho \dashv \Sigma$. That is, a choreography $\Sigma = (I_1 : \sigma_1 \mid \cdots \mid I_n : \sigma_n) \setminus L$ is a (complex) service whose interface is $\mathtt{int}(\Sigma) \stackrel{\text{def}}{=} \bigcup_{1 \leq i \leq n} I_i \setminus L$ and whose behavior is the combination of the behaviors of the participants running in parallel. In this setting, a (client) contract $J : \rho$ *is compliant with* the choreography Σ if $J \subseteq \mathtt{int}(\Sigma)$ and $\rho \dashv \Sigma$.

Definition 6 (Choreography refinement). *Let $\Sigma = (I_1 : \sigma_1 \mid \cdots \mid I_n : \sigma_n) \setminus L$ and $\Sigma' = (I'_1 : \sigma'_1 \mid \cdots \mid I'_n : \sigma'_n) \setminus L'$. The choreography Σ' is a* refinement *of Σ if and only if the following conditions hold:*

1. $\mathtt{int}(\Sigma) = \mathtt{int}(\Sigma')$*;*
2. *for every $1 \leq i \leq n$ we have $I_i : \sigma_i \preceq I'_i : \sigma'_i$;*
3. *for every $1 \leq i, j \leq n$ with $i \neq j$ we have that $L' \cap (\mathtt{names}(\sigma'_i) \setminus \mathtt{names}(\sigma_i)) \cap (\mathtt{names}(\sigma'_j) \setminus \mathtt{names}(\sigma_j)) = \emptyset$.*

Refinement defines a "safe" replacement of activities in a choreography with more detailed ones (such as their implementations) still preserving the overall interface of the choreography (condition 1). The replacing activities may have more capabilities than those offered by the replaced ones (condition 2) and it must not be the case that the new activities communicate with each other by means of names that do not occur in the original choreography (condition 3) for otherwise the original choreography specification would be violated.

We now prove a soundness result for the notion of refinement which represents a restricted form of precongruence of $\preceq$ with respect to the parallel composition. The result is not based on any particular property (e.g. deadlock freedom) of the choreography itself. We merely show that, from the point of view of a client interacting with a choreography as a whole, the refinement of the choreography does not jeopardize the completion of the client.

Theorem 4. *Let $I : \rho$ be compliant with a choreography Σ_1 and Σ_2 be a refinement of Σ_1. Then $I : \rho$ is also compliant with Σ_2.*

6 Concluding Remarks

In this contribution we have studied a formal theory of Web services contracts. The subcontract preorder used in the theory arises semantically as in the testing setting, except that the notion of "passing a test" is essentially different and reflects more faithfully the interaction of clients and services. We have given two different characterizations of the subcontract preorder, one directly induced by the notion of compliance, the other one that is more amenable for an algorithmic implementation. The subcontract relation is effectively and efficiently applicable in any query-based system for service discovery because it is supported by a notion of principal dual contract. It is also applicable in choreographies for replacing contracts with larger ones.

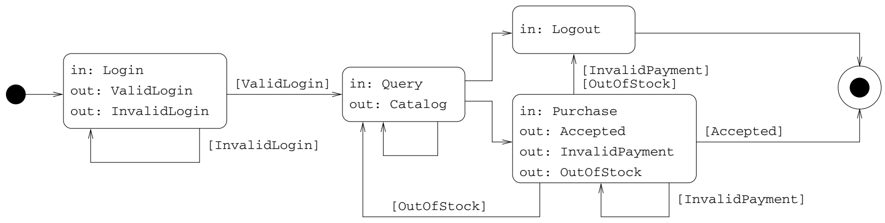


Fig. 1. Contract of a simple e-commerce service as a WSCL diagram

The theory may be used as a foundation of Web services technologies, such as WSDL and WSCL. In [6] we already discussed how to encode WSDL message-exchange patterns and acyclic WSCL diagrams into the recursion-free fragment of the contract language. The language in this paper allows us to express also cyclic WSCL conversations by means of recursion. Figure 1 shows a WSCL diagram describing the protocol of a service requiring clients to login before they can issue a query. After the query, the service returns a catalog. From this point on, the client can decide whether to purchase an item from the catalog or to logout and leave. In case of purchase, the service may either report that the purchase was successful, or that the item is out-of-stock, or that the client's payment was refused. By interpreting names as message types, this e-commerce service can be encoded in our language by a contract whose behavior is:

$$\begin{array}{l}\mathsf{rec}\ x.\mathtt{Login}.(\overline{\mathtt{InvalidLogin}}.x \oplus \overline{\mathtt{ValidLogin}}.\mathsf{rec}\ y.\\ \quad \mathtt{Query}.\overline{\mathtt{Catalog}}.(y + \mathtt{Logout} + \mathsf{rec}\ z.\mathtt{Purchase}.\\ \quad\quad \overline{\mathtt{Accepted}} \oplus \overline{\mathtt{InvalidPayment}}.(z + \mathtt{Logout}) \oplus \overline{\mathtt{OutOfStock}}.(y + \mathtt{Logout})))\end{array}$$

We notice the correspondence between unlabeled (respectively, labeled) transitions in Figure 1 and external (respectively, internal) choices in the contract. We also notice how recursion is used for expressing iteration (the cycles in the figure) so that the client is given another chance whenever an action fails for some reason.

Several future research directions stem from this work. On the technical side, we would like to define and investigate an axiomatic characterization of $\preceq$. We expect such axiomatization to closely resemble the one for the $\sqsubseteq_{\mathtt{must}}$ preorder [19,15]. On the linguistic side we would like to explore new process constructions that could take into account information available with contracts. For instance, imagine a client that wants to use a service exporting the contract $a \oplus b$; in the simple language of Section 2 the client cannot specify that it wants to connect on b if available, and on a otherwise, for the choice operators are symmetric. It is unclear to which extent such constructs affect the $\preceq$ preoder over contracts. The discussion of Proposition 2 suggests that there are interesting connections between the properties of the $\preceq$ preorder and the boolean operators over sets of compliant clients. We aim to further explore such set-theoretic interpretation of contracts and to devise a query language for service discovery that provides primitive operators for union, intersection, and negation for contracts. The authors of [6] provide a type system to extract the contract out of a recursion-free fragment of CCS-like process calculus. We aim to extend such type system to full CCS, although the task is not trivial because CCS processes may exhibit a non-regular behavior. When the behavior cannot be captured accurately by our contract language, regular under- and over-estimations must be provided. Finally, a major task is to move our investigation from a CCS-like formalism to a π-calculus one. This will allow us to generalize the forthcoming version of WSDL which enables the possibility to describe higher-order Web services, but also to model callbacks, continuation-passing and dynamically bound Web services.

References

1. Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., Liu, C.K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A.: Web Services Business Process Execution Language Version 2.0 (January 2007), http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.html
2. Banerji, A., Bartolini, C., Beringer, D., Chopella, V., et al.: Web Services Conversation Language (WSCL) 1.0 (March 2002), http://www.w3.org/TR/2002/NOTE-wscl10-20020314

3. Beringer, D., Kuno, H., Lemon, M.: Using WSCL in a UDDI Registry 1.0, UDDI Working Draft Best Practices Document (2001), http://xml.coverpages.org/HP-UDDI-wscl-5-16-01.pdf
4. Bravetti, M., Zavattaro, G.: Towards a unifying theory for choreography conformance and contract compliance. In: Pre-proceedings of 6th Symposium on Software Composition (2007)
5. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centered programming for web services. In: Proceedings of 16th European Symposium on Programming, LNCS, Springer, Heidelberg (2007)
6. Carpineti, S., Castagna, G., Laneve, C., Padovani, L.: A formal account of contracts for Web Services. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 148–162. Springer, Heidelberg (2006)
7. Castagna, G., Gesbert, N., Padovani, L.: A theory of contracts for Web Services. In: Proceedings of 5th ACM SIGPLAN Workshop on Programming Language Technologies for XML, pp. 37–48. ACM Press, New York (2007)
8. Chaki, S., Rajamani, S.K., Rehof, J.: Types as models: model checking message-passing programs. SIGPLAN Not. 37(1), 45–57 (2002)
9. Chinnici, R., Haas, H., Lewis, A.A., Moreau, J.-J., et al.: Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts (March 2006), http://www.w3.org/TR/2006/CR-wsdl20-adjuncts-20060327
10. Chinnici, R., Moreau, J.-J., Ryman, A., Weerawarana, S.: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language (March 2006), http://www.w3.org/TR/2006/CR-wsdl20-20060327
11. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1 (2001), http://www.w3.org/TR/2001/NOTE-wsdl-20010315
12. Colgrave, J., Januszewski, K.: Using WSDL in a UDDI registry, version 2.0.2. Technical note, OASIS (2004), http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-w sdl-v2.htm
13. Gay, S., Hole, M.: Subtyping for session types in the π-calculus. Acta Informatica 42(2-3), 191–225 (2005)
14. Hennessy, M.: Acceptance trees. JACM: Journal of the ACM 32(4), 896–928 (1985)
15. Hennessy, M.C.B.: Algebraic Theory of Processes. Foundation of Computing, MIT Press, Cambridge (1988)
16. Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., Barreto, C.: Web Services Choreography Description Language 1.0 (2005), http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/
17. Milner, R.: A Calculus of Communicating Systems. Springer, Heidelberg (1982)
18. Nicola, R.D., Hennessy, M.: Testing equivalences for processes. Theor. Comput. Sci 34, 83–133 (1984)
19. Nicola, R.D., Hennessy, M.: CCS without τ's. In: Ehrig, H., Levi, G., Montanari, U. (eds.) CAAP 1987 and TAPSOFT 1987. LNCS, vol. 249, pp. 138–152. Springer, Heidelberg (1987)
20. Nielson, H.R., Nielson, F.: Higher-order concurrent programs with finite communication topology (extended abstract). In: POPL '94. Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 84–97. ACM Press, New York, NY, USA (1994)
21. Parastatidis, S., Webber, J.: MEP SSDL Protocol Framework (April 2005), http://ssdl.org

Topology-Dependent Abstractions of Broadcast Networks

Sebastian Nanz, Flemming Nielson, and Hanne Riis Nielson

Informatics and Mathematical Modelling
Technical University of Denmark
{nanz,nielson,riis}@imm.dtu.dk

Abstract. Broadcast semantics poses significant challenges over point-to-point communication when it comes to formal modelling and analysis. Current approaches to analysing broadcast networks have focused on fixed connectivities, but this is unsuitable in the case of wireless networks where the dynamically changing network topology is a crucial ingredient. In this paper we develop a static analysis that automatically constructs an abstract transition system, labelled by actions and connectivity information, to yield a mobility-preserving finite abstraction of the behaviour of a network expressed in a process calculus with asynchronous local broadcast. Furthermore, we use model checking based on a 3-valued temporal logic to distinguish network behaviour which differs under changing connectivity patterns.

1 Introduction

Broadcast communication, in contrast to point-to-point message passing, is employed in a wide range of networking paradigms such as Ethernet and wireless LAN, mobile telephony, or mobile ad-hoc networks. These can be further distinguished into approaches where broadcast is taken to be global, i.e. all nodes of the network receive a broadcast message, or local, such that only neighbours of the broadcasting node are able to receive. In order to obtain a formal model for the latter case, the network topology has to be encoded by the chosen modelling formalism to express the notion of a neighbourhood. Furthermore, the connectivity may change over time, caused by node mobility or similar changes in environment conditions which are not controlled by the nodes' protocol actions.

This mix of broadcast behaviour and mobility has turned out to be a challenge for automated verification and analysis techniques. For instance, model checking of mobile ad-hoc networks, in a line of work started by [2], has remained limited to fixed connectivities. In our previous work on static analysis of mobile ad-hoc networks [11], topology changes are considered in the modelling, but abstracted into a fixed representation for the sake of the analysis, hence achieving a safe description of the network, but losing the ability to expose network behaviour related to connectivity change.

In this paper we address these deficiencies by defining *abstract transition systems* which provide finite abstractions of the behaviour of broadcast networks specified in the broadcast calculus bKlaim, which is also introduced in this paper. The abstractions preserve mobility in the sense that their transitions depend on connectivity information, and hence reflect changes in connectivity. We present a 3-valued interpretation of formulae of Action Computation Tree Logic (ACTL) [13] on abstract transition systems, which captures the nature of the abstraction by evaluating to "unknown" whenever the abstraction prevents definite conclusions about the concrete behaviour of the related bKlaim network.

We also show how abstract transition systems can be algorithmically constructed from networks specified in bKlaim. This is done using a static analysis, based on the idea of Monotone Frameworks [14], which also gives us fine-grained control over the coarseness of the abstraction. This analysis has been implemented, and we show how the complete framework enables us expose the influence of the network dynamics on the resulting network state.

The remainder of the paper is structured as follows. In §2 we present the syntax and operational semantics of bKlaim. In §3 we introduce abstract transition systems, and describe 3-valued ACTL and its relation to the concrete transition system of bKlaim. We develop a Monotone Framework and worklist algorithm to construct abstract transition systems for bKlaim networks in §4. We conclude in §5. A technical report [12] contains full proofs.

2 bKlaim

Process calculi of the Klaim family [1] are centred around the *tuple space* paradigm in which a system is comprised by a distributed set of nodes that communicate by placing tuples into and getting tuples from one or more shared tuple spaces. In this paper we use this basic paradigm to model systems communicating via *local broadcast*, i.e. only nodes within the neighbourhood of the broadcasting node may receive a sent message tuple; this distinguishes bKlaim from the broadcast calculus CBS [19], where all broadcast is global. In contrast to the standard Klaim semantics, where tuple spaces are shared resources among all nodes, we instrument this approach for the modelling of local broadcast: broadcast messages are output into the tuple spaces of neighbouring nodes to the sending node, where they can be picked up only by the processes residing at the respective locations; this yields an *asynchronous* version of local broadcast, in contrast to the calculi CBS$^\sharp$ [11] and CMN [9] which both feature synchronous behaviour. The notion of neighbourhood is expressed by *connectivity graphs*, which specify the locations currently connected with a sender and may change during the evolution of the network.

2.1 Syntax

The bKlaim calculus comprises three parts: networks, processes, and actions. Networks give the overall structure in which processes and tuple spaces are located, and processes execute by performing actions. An overview of the syntax is shown in Table 1.

Table 1. Syntax of a fragment of bKlaim

$N ::=$	$l :: P$	located node	$a^\ell ::=$	$\mathsf{bcst}^\ell(t)$	broadcast output
	$\mid\ l :: S$	located tuple space		$\mid\ \mathsf{out}^\ell(t)$	output
	$\mid\ N_1 \parallel N_2$	net composition		$\mid\ \mathsf{in}^\ell(T)$	input
$P ::=$	nil	null process	$T ::=$	$F \mid F, T$	templates
	$\mid\ a^\ell.P$	action prefixing	$F ::=$	$f \mid\ !x$	template fields
	$\mid\ P_1 \mid P_2$	parallel composition	$t ::=$	$f \mid f, t$	tuples
	$\mid\ A$	process invocation	$f ::=$	$v \mid l \mid x$	tuple fields

Tuples are finite lists of tuple fields, which comprise values $v \in \mathbf{Val}$, locations $l \in \mathbf{Loc}$, and variables $x \in \mathbf{Var}$. We assume in general that locations are just distinguished values, i.e. $\mathbf{Loc} \subseteq \mathbf{Val}$. A ground tuple t is an element of $\mathbf{Val}^*$. *Templates* are used as patterns to select tuples in a tuple space. They are finite lists of tuple fields and formal fields $!x$ which are used to bind variables to values; within a template, x must not occur both as a variable and a formal field, or in more than one formal field. The set $fv(t)$ containing the free variables of tuple t are defined as usual, and the definition of fv can be extended to templates, actions, and processes. Values are free as there are no binding statements for them.

Networks consist of located processes and tuple spaces. In contrast to Klaim, a tuple space S is taken to be a multiset (rather than a set) of tuples, i.e. a total map from the set of tuples into $\mathbb{N}_0$. We say that a tuple t is in the domain $dom(S)$ of S if $S(t) > 0$, and use the following notation to express that a copy of tuple t is added to or removed from a multiset S:

$$S[t]^\uparrow = \lambda u. \begin{cases} S(u) + 1 & \text{if } u = t \\ S(u) & \text{otherwise} \end{cases} \qquad S[t]^\downarrow = \lambda u. \begin{cases} S(u) - 1 & \text{if } u = t \wedge S(u) > 0 \\ S(u) & \text{otherwise} \end{cases}$$

We also introduce below a well-formedness condition which ensures that there is exactly one tuple space per location. This is because tuple spaces in bKlaim are not seen as freely shared among nodes, but as private components (stores) associated with the processes residing at the same location.

A *process* is either the terminated process nil, a process prefixed with an action to be executed, a parallel composition, or a process invocation to express recursive behaviour. Process definitions are of the form $A \triangleq P$, where P is closed, i.e. contains no free variables. As an abbreviation, we may sometimes use the notation $A(t) \triangleq P$ and have P parameterised in the free variables of t.

Actions are equipped with labels $\ell \in \mathbf{Lab}$ which facilitate the analysis in §4. The action $\mathsf{bcst}^\ell(t)$ places a tuple t into the set of tuple spaces belonging to the current neighbours of the sending node, thus describing local broadcast. Neighbourhoods are defined at the semantic level via the notion of connectivity graphs. The action $\mathsf{out}^\ell(t)$ models the output of a tuple to the private tuple space of the node performing this action. Using $\mathsf{in}^\ell(T)$, processes retrieve tuples which match the template T from their private tuple space and remove it. Note that there is no statement corresponding to Klaim's creation of new locations $\mathsf{newloc}(l)$ because we want to deal with a given set of located nodes which cannot spawn themselves by process actions. Because of space constraints, some actions contained in the full version of bKlaim (such as process migration) are omitted in this description. We refer the reader to the companion technical report [12].

Example 1. We describe a simple protocol for information retrieval in mobile ad-hoc networks. A mobile ad-hoc network is a special kind of wireless network, where participating nodes form temporary multi-hop connections and may act as both host and router, i.e. both sending own requests and relaying messages for others. The protocol is specified in bKlaim as follows:

$$\begin{aligned}
&\mathit{Snd}(x) &&\triangleq \mathsf{bcst}^1(\mathtt{ask}, x).\mathit{Rec}(x)\\
&\mathit{Rec}(x) &&\triangleq \mathsf{in}^2(\mathtt{has}, !l, x, !y).\mathit{Rec}(x)\\
&\mathit{Prc}(l) &&\triangleq \mathsf{in}^3(\mathtt{ask}, !x).(\mathsf{in}^4(x, !y).\mathsf{bcst}^5(\mathtt{has}, l, x, y) \mid \mathsf{bcst}^6(\mathtt{ask}, x).\mathit{Prc}(l))\\
&\mathit{Rel} &&\triangleq \mathsf{in}^7(\mathtt{has}, !l, !x, !y).\mathsf{bcst}^8(\mathtt{has}, !l, x, y).\mathit{Rel}\\
\\
&\mathit{Net} &&\triangleq \mathtt{l}_1 :: \mathit{Snd}(\mathtt{t}) \parallel \mathtt{l}_2 :: (\mathit{Prc}(\mathtt{l}_2) \mid \mathit{Rel}) \parallel \mathtt{l}_2 :: [[\mathtt{t}, \mathtt{i}_2] \mapsto 1]\\
& &&\quad \parallel \mathtt{l}_3 :: (\mathit{Prc}(\mathtt{l}_3) \mid \mathit{Rel}) \parallel \mathtt{l}_3 :: [[\mathtt{t}, \mathtt{i}_3] \mapsto 1]
\end{aligned}$$

The protocol is initiated on network *Net* when node $\mathtt{l}_1$ executes the process *Snd* to search for information on topic $\mathtt{t}$. Node $\mathtt{l}_1$ then enters a state where it waits for (possibly multiple) answers of the form $(\mathtt{has}, l, x, y)$, meaning that the node at location l sent content y concerning topic x.

Nodes $\mathtt{l}_2$ and $\mathtt{l}_3$ can process ask-messages using *Prc*. Upon reception, each of the nodes check whether they have content available in their tuple spaces which match topic x. If so, they broadcast a has-message containing this content. In order to make sure that the ask-message is propagated across the whole of the network, they also rebroadcast this message, and restart process *Prc* to be ready to receive other requests.

Rel is a simple relay process for has-messages. Note further that $\mathtt{l}_2$ and $\mathtt{l}_3$ have tuple spaces with contents $\mathtt{i}_2$ and $\mathtt{i}_3$ associated with topic $\mathtt{t}$.

2.2 Operational Semantics

As a prerequisite for defining the operational semantics of bKlaim, we have to give a notion of connectivity between nodes. A *connectivity graph* as in [11,10] is a directed graph G on a subset of the set of locations **Loc**. As usual, $V(G)$ denotes the set of vertices of G and $E(G)$ its set of edges. Given a graph G, we write

$$G(l) = \{l' \ : \ (l, l') \in E(G)\}$$

to denote the *neighbourhood* of a location l.

In this way, a connectivity graph G gives a straightforward notion of connectivity to a network N: a node at location l' may receive a message sent by a node at location l if and only if $(l, l') \in E(G)$. Because the graph is directed, both unidirectional and bidirectional links can be expressed. Note that by separating connectivity from process actions (which most readily distinguishes bKlaim from the bπ-calculus [5] for example) we are able to express the behaviour of a variety of networks in which the connectivity may change through changes in the environment conditions, which are not expressed by process actions. Wireless networks are one example, where node movements trigger both link failures and the establishment of new links.

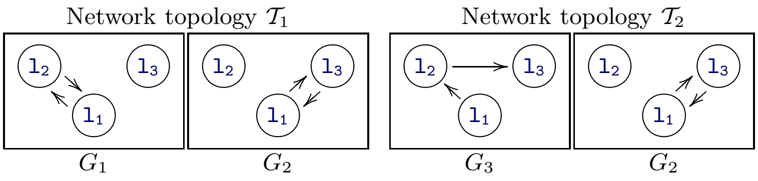
Connectivity graphs provide a snapshot of the network connectivity. In contrast, a *network topology* $\mathcal{T}$ is a set of connectivity graphs which share the same set of vertices. We use network topologies to express the set of possible configurations a particular network may be in.

In order to ensure that a network topology and a network agree, we introduce a well-formedness condition. We first extend the definition of the vertex function V from graphs to networks:

$$V(l :: P) = V(l :: S) = \{l\} \quad \text{and} \quad V(N_1 \parallel N_2) = V(N_1) \cup V(N_2)$$

We say that the pair $(N, \mathcal{T})$ of a network N and network topology $\mathcal{T}$ is *well-formed* if there is exactly one located tuple space $l :: S$ for each $l \in V(N)$, and if furthermore $\mathcal{T}$ contains only connectivity graphs G with $V(G) = V(N)$.

Example 2. Continuing Example 1, we define the following network topologies over $V(Net)$:



Network topology $\mathcal{T}_1$
Network topology $\mathcal{T}_2$
$\mathtt{l}_1$
$\mathtt{l}_2$
$\mathtt{l}_3$
G_1
G_2
G_3
G_2

We give the operational semantics of bKlaim by a *reduction relation* of the form $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}}_G N$, defined in Table 2, together with a straightforward *structural congruence* $M \equiv N$ and *template matching* semantics in Table 3. Derivations of a network N via the reduction relation are with respect to a network topology $\mathcal{T}$ where $(N, \mathcal{T})$ are well-formed; the operational semantics ensures that well-formedness is preserved over all derivations. A derivation is parametrised with a connectivity graph $G \in \mathcal{T}$ to express that the derivation holds under the connectivity expressed by G. We may drop the parameter G and write $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}} N$ when a transition does not depend on the actual choice of $G \in \mathcal{T}$. For the sake of the analysis in §4, transitions are labelled with labels $\mathbb{l}$ of the form (l, ℓ) and $(l, \ell[t])$, to express that the action labelled ℓ has executed at location l, and – in the case of the in^ℓ-action only – that the tuple t has been input at location l.

The bcst-rule puts a tuple t into all tuple spaces in the current neighbourhood $G(l)$ of the sender location l, where the current neighbourhood is nondeterministically chosen from the network topology $\mathcal{T}$. Rule out puts a tuple t into the private tuple space at location l. The in-rule inputs (deletes) a tuple contained in the private tuple space S if it matches to the template T, and continues with the process $P\sigma$, where σ captures the bindings introduced by the template matching.

Table 2. Reduction relation of bKlaim

$$\frac{G \in \mathcal{T}}{\mathcal{T} \vdash l :: \mathsf{bcst}^{\ell}(t).P \parallel \prod_{l' \in G(l)} l' :: S_{l'} \xrightarrow{(l,\ell)}_G l :: P \parallel \prod_{l' \in G(l)} l' :: S_{l'}(t)^{\uparrow}}$$

$$\frac{}{\mathcal{T} \vdash l :: \mathsf{out}^{\ell}(t).P \parallel l :: S \xrightarrow{(l,\ell)} l :: P \parallel l :: S(t)^{\uparrow}}$$

$$\frac{S(t) > 0 \qquad match(T,t) = \sigma}{\mathcal{T} \vdash l :: \mathsf{in}^{\ell}(T).P \parallel l :: S \xrightarrow{(l,\ell[t])} l :: P\sigma \parallel l :: S(t)^{\downarrow}}$$

$$\frac{\mathcal{T} \vdash M \xrightarrow{\mathbb{1}} M'}{\mathcal{T} \vdash M \parallel N \xrightarrow{\mathbb{1}} M' \parallel N} \qquad \frac{N \equiv M \quad \mathcal{T} \vdash M \xrightarrow{\mathbb{1}} M' \quad M' \equiv N'}{\mathcal{T} \vdash N \xrightarrow{\mathbb{1}} N'}$$

Table 3. Structural congruence and template matching of bKlaim

$$\begin{aligned} N_1 \parallel N_2 &\equiv N_2 \parallel N_1 \\ (N_1 \parallel N_2) \parallel N_3 &\equiv N_1 \parallel (N_2 \parallel N_3) \\ l :: P &\equiv l :: P \mid \mathsf{nil} \\ l :: A &\equiv l :: P \text{ if } A \triangleq P \\ l :: P_1 \mid P_2 &\equiv l :: P_1 \parallel l :: P_2 \end{aligned}$$

$$match(v,v) = \epsilon \qquad match(!x,v) = [v/x]$$

$$\frac{match(F,f) = \sigma_1 \qquad match(T,t) = \sigma_2}{match((F,T),(f,t)) = \sigma_1 \circ \sigma_2}$$

3 Abstract Transition Systems

For a given network, the operational semantics of bKlaim gives rise to a (possibly infinite) transition system where the transitions are determined by the actions performed at each step and the connectivity the network has to abide by when performing a step. For the sake of analysis, we are interested in transforming this transition system into a *finite* one which still preserves the influence of the network topology on the resulting network states. For this purpose this section introduces *abstract transition systems*, and a version of Action Computation Tree Logic (ACTL) [13] to describe their properties. In order to accommodate the notion of abstraction in the logic, we use a 3-valued interpretation on abstract transition systems so that a formula evaluates to "unknown" whenever the abstraction prevents us from obtaining a definite result; when evaluating to "true" or "false" however, an embedding theorem ensures that the same formula holds (resp. fails) in its 2-valued interpretation on the concrete transition system.

3.1 Exposed Actions

This section introduces the notion of exposed actions which is used to express abstract network configurations; abstract transition systems, introduced in the following section, will then describe transitions between such abstract configurations, which are related to transitions between concrete networks.

An *exposed action* is an action (or tuple) that *may* participate in the next interaction. In general, a process may contain many, even infinitely many, occurrences of the same action (all identified by the same label) and it may be that several of them are ready to participate in the next interaction. To capture this, we define an *extended multiset* M as an element of:

$$\mathfrak{M} = \mathbf{Loc} \times (\mathbf{Lab} \cup \mathbf{Val}^*) \to \mathbb{N} \cup \{\infty\}$$

The idea is that $M(l, \ell)$ records the number of occurrences of the label ℓ, and analogously $M(l, t)$ the number of occurrences of the tuple t, at a location l; there may be a finite number, in which case $M(\mathbb{l}) \in \mathbb{N}$, or an infinite number, in which case $M(\mathbb{l}) = \infty$ (where $\mathbb{l}$ ranges over (l, ℓ) or (l, t)). The set $\mathfrak{M}$ is equipped with a partial ordering $\leq_{\mathfrak{M}}$ defined by:

$$M \leq_{\mathfrak{M}} M' \text{ iff } \forall\, \mathbb{l}.\ M(\mathbb{l}) \leq M'(\mathbb{l}) \vee M'(\mathbb{l}) = \infty$$

The domain $(\mathfrak{M}, \leq_{\mathfrak{M}})$ is a complete lattice, and in addition to least and greatest upper bound operators, we shall need operations $+_{\mathfrak{M}}$ and $-_{\mathfrak{M}}$ for addition and subtraction, which can be defined straightforwardly.

To calculate exposed actions, we shall introduce the function $\mathcal{E} : \mathbf{Net} \to \mathfrak{M}$ which takes a network and calculates its extended multiset of exposed actions; this function is defined as follows:

$$\begin{aligned}
\mathcal{E}[\![N_1 \parallel N_2]\!] &= \mathcal{E}[\![N_1]\!] +_{\mathfrak{M}} \mathcal{E}[\![N_2]\!] & \mathcal{E}_l[\![\mathsf{nil}]\!] &= \bot_{\mathfrak{M}} \\
\mathcal{E}[\![l :: P]\!] &= \mathcal{E}_l[\![P]\!] & \mathcal{E}_l[\![a^{\ell}.P]\!] &= \bot_{\mathfrak{M}}[(l, \ell) \mapsto 1] \\
\mathcal{E}[\![l :: S]\!] &= \textstyle\sum_{\mathfrak{M}, t} \bot_{\mathfrak{M}}[(l, t) \mapsto S(t)] & \mathcal{E}_l[\![P_1 \mid P_2]\!] &= \mathcal{E}_l[\![P_1]\!] +_{\mathfrak{M}} \mathcal{E}_l[\![P_2]\!] \\
& & \mathcal{E}_l[\![A]\!] &= \mathcal{E}_l[\![P]\!] \text{ if } A \triangleq P
\end{aligned}$$

Note that in the case for tuple spaces, every tuple $t \in S$ is recorded with according multiplicity $S(t)$ at location l. In the case of actions $a^{\ell}.P$, the label ℓ is recorded at location l with multiplicity 1. The remaining cases are straightforward. The operations involved in the definition of $\mathcal{E}$ are all monotonic such that a least fixed point is ensured by Tarski's fixed point theorem.

Example 3. Continuing Example 1, it is easy to check that

$$\begin{aligned}
\mathcal{E}[\![Net]\!] = [&(\mathtt{l}_1, 1) \mapsto 1, (\mathtt{l}_2, 3) \mapsto 1, (\mathtt{l}_2, 7) \mapsto 1, (\mathtt{l}_3, 3) \mapsto 1, \\
&(\mathtt{l}_3, 7) \mapsto 1, (\mathtt{l}_2, [\mathtt{t}, \mathtt{i}_2]) \mapsto 1, (\mathtt{l}_3, [\mathtt{t}, \mathtt{i}_3]) \mapsto 1].
\end{aligned}$$

We can show that the exposed actions are invariant under the structural congruence and that they correctly capture the actions that may be involved in the first reduction step.

Lemma 1. *If $M \equiv N$, then $\mathcal{E}[\![M]\!] = \mathcal{E}[\![N]\!]$. Furthermore, if $\mathcal{T} \vdash M \xrightarrow{\mathbb{1}}_G N$ and $\mathbb{1} = (l, \ell)$, then $\mathbb{1} \in dom(\mathcal{E}[\![M]\!])$; and if $\mathbb{1} = (l, \ell[t])$, then $(l, \ell), (l, t) \in dom(\mathcal{E}[\![M]\!])$.*

3.2 Abstract Transition Systems

An *abstract transition system* is a quadruple $(\mathsf{Q}, q_0, \delta, \mathsf{E})$ with the following components: A finite set of states Q where each state q is associated with an extended multiset $\mathsf{E}[q]$ and the idea is that q represents all networks N with $\mathcal{E}[\![N]\!] \leq_{\mathfrak{M}} \mathsf{E}[q]$; an initial state q_0, representing the initial network N_0; a finite transition relation δ, where $(q_s, (G, \mathbb{1}), q_t) \in \delta$ reflects that starting in state q_s, under connectivity G, the action $\mathbb{1}$ may execute and give rise to q_t.

Definition 1. *We say that a state denoting the multiset E* represents *a network N, written $N \rhd E$, iff $\mathcal{E}[\![N]\!] \leq_{\mathfrak{M}} E$.*

Definition 2. *We say that an abstract transition system* $(\mathsf{Q}, q_0, \delta, \mathsf{E})$ faithfully describes *the evolution of a network N_0 if:*

$$M \rhd \mathsf{E}[q_s] \text{ and } \mathcal{T} \vdash N_0 \rightarrow^* M \xrightarrow{\mathbb{l}}_G N,$$

imply that there exists a unique $q_t \in \mathsf{Q}$ such that

$$N \rhd \mathsf{E}[q_t] \text{ and } (q_s, (G, \mathbb{l}), q_t) \in \delta.$$

In §4 we shall show how to construct an abstract transition system that faithfully describes the evolution of a given network N.

Example 4. For the network $(Net, \mathcal{T}_1)$ of Example 1, the static analysis of §4 generates an abstract transition system with 27 states and 46 transitions. We look at one of these transitions in detail, namely $(q_3, (G_1, (\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2])), q_6) \in \delta$. For the states q_3 and q_6 involved in this transition, it holds that

$$dom(\mathsf{E}[q_3]) = \{(\mathtt{l}_1, 2), (\mathtt{l}_2, 4), (\mathtt{l}_2, 6), (\mathtt{l}_2, 7), (\mathtt{l}_3, 3), (\mathtt{l}_3, 7), (\mathtt{l}_2, [\mathtt{t}, \mathtt{i}_2]), (\mathtt{l}_3, [\mathtt{t}, \mathtt{i}_3])\}$$
$$dom(\mathsf{E}[q_6]) = \{(\mathtt{l}_1, 2), (\mathtt{l}_2, 5), (\mathtt{l}_2, 6), (\mathtt{l}_2, 7), (\mathtt{l}_3, 3), (\mathtt{l}_3, 7), (\mathtt{l}_3, [\mathtt{t}, \mathtt{i}_3])\}$$

and therefore state q_3 represents a network of the form

$$\mathtt{l}_1 :: \mathsf{in}^2(...).Rec(\mathtt{t}) \parallel \mathtt{l}_2 :: (\mathsf{in}^4(...).\mathsf{bcst}^5(...) \ldots \mid \mathsf{bcst}^6(...).Prc(\mathtt{l}_2)) \parallel \mathtt{l}_2 :: [(\mathtt{t}, \mathtt{i}_2) \mapsto 1] \parallel \ldots$$

and after a transition under connectivity graph G_1 with action $(\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2])$ (and analogously for G_2, as label 4 denotes a (local) input action which thus does not depend on connectivity), we end up in state q_6 that represents

$$\mathtt{l}_1 :: \mathsf{in}^2(...).Rec(\mathtt{t}) \parallel \mathtt{l}_2 :: (\mathsf{bcst}^5(...). \ldots \mid \mathsf{bcst}^6(...).Prc(\mathtt{l}_2)) \parallel \mathtt{l}_2 :: [(\mathtt{t}, \mathtt{i}_2) \mapsto 0] \parallel \ldots.$$

3.3 Interpretation of ACTL Properties

In order to express properties about a network, we are using a variant of Action Computation Tree Logic (ACTL) [13], which allows us to utilise the labels $(G, \mathbb{l})$ on the edges of an abstract transition system to constrain the set of paths we are interested in; in this way we may for example determine which properties hold if only node movements specified by a subset $\mathcal{T}' \subseteq \mathcal{T}$ of the original topology are considered. The following grammar describes the syntax of path formulae ϕ and state formulae γ:

$$\phi ::= \mathtt{tt} \mid \mathbb{ll} \mid \neg\phi \mid \phi \wedge \phi \mid \exists\gamma$$
$$\gamma ::= \mathbf{X}_{\Omega}\, \phi \mid \phi\, \mathbf{U}_{\Omega}\, \phi$$

Here, $\mathbb{ll}$ denotes (l, ℓ) or (l, t), $\exists$ is a path quantifier, Ω is a set of transition labels $(G, \mathbb{l})$ and will be used to constrain the paths a formula is evaluated on, and $\mathbf{X}_\Omega$ and $\mathbf{U}_\Omega$ are *next* and *until* operators, respectively. We shall give two interpretations of this logic; the first relates to the concrete semantics of §2.

We define two judgements $N \vDash \phi$ and $\Pi \vDash \gamma$ for satisfaction of ϕ by a network N, and γ by a path Π. A path Π is of the form $(N_0, (G_0, \mathbb{l}_0), N_1, (G_1, \mathbb{l}_1), \dots)$ where $\Pi(i) \xrightarrow{\mathbb{l}_i}_{G_i} \Pi(i+1)$ for $i \geq 0$ (we write $\Pi(i)$ for N_i, and $\Pi[i]$ for $(G_i, \mathbb{l}_i)$).

$$\begin{array}{lll@{\qquad}lll}
N \vDash \mathtt{tt} & & & N \vDash \mathbb{l} & \text{iff} & \mathbb{l} \in \mathcal{E}[\![N]\!] \\
N \vDash \neg\phi & \text{iff} & N \nvDash \phi & N \vDash \phi_1 \wedge \phi_2 & \text{iff} & N \vDash \phi_1 \wedge N \vDash \phi_2 \\
N \vDash \exists\gamma & \text{iff} & \multicolumn{4}{l}{\text{there exists a path } \Pi \text{ such that } \Pi(0) = N \text{ and } \Pi \vDash \gamma} \\
\Pi \vDash \mathbf{X}_\Omega\, \phi & \text{iff} & \multicolumn{4}{l}{\Pi(1) \vDash \phi \text{ and } \Pi[0] \in \Omega} \\
\Pi \vDash \phi_1\, \mathbf{U}_\Omega\, \phi_2 & \text{iff} & \multicolumn{4}{l}{\text{there exists } k \geq 0 \text{ such that } \Pi(k) \vDash \phi_2 \text{ and for all } 0 \leq i < k :} \\
 & & \multicolumn{4}{l}{\Pi(i) \vDash \phi_1 \text{ and } \Pi[i] \in \Omega}
\end{array}$$

Thus the semantics of formulae closely resembles that of ACTL, with the exception that for the novel clause $\mathbb{l}$ to evaluate to satisfy network N, $\mathbb{l}$ must be exposed in N.

Clearly, we cannot directly establish satisfaction of a formula on a network because the related transition system might be infinite. We therefore propose to check formulae on the basis of abstract transition systems, and formally relate the results obtained to the concrete network evolution.

The important question is how to represent the nature of the abstraction. A natural way to model the uncertainty of whether an abstract edge is present in the concrete transition system is to use a 3-valued logic. Here the classical set of truth values $\{0, 1\}$ is extended with a value $1/2$ for expressing the uncertainty. Several choices of 3-valued logics exist and we choose here to use Kleene's strongest regular 3-valued logic [7]; this is in line with the developments of [3,20]. Formulae defined over the abstraction may make use of all three truth values, but unlike e.g. [20,15], the abstraction itself will only make use of the value 0 and $1/2$.

A simple way to define conjunction (resp. disjunction) in this logic is as the minimum (resp. maximum) of its arguments, under the order $0 < 1/2 < 1$. We write *min* and *max* for these functions, and extend them to sets in the obvious way, with $\mathit{min}\,\emptyset = 1$ and $\mathit{max}\,\emptyset = 0$. Negation $\neg^3$ maps 0 to 1, 1 to 0, and $1/2$ to $1/2$. Other operations can be lifted from the classical setting to the 3-valued setting using the method of [16].

Let $L(q, \mathbb{l}) = 0$ if $\mathbb{l} \notin \mathsf{E}[q]$, and $1/2$ otherwise. Furthermore, let $D_\Omega(q_s, q_t) = 0$ if $(q_s, (G, \mathbb{l}), q_t) \notin \delta$ for all $(G, \mathbb{l}) \in \Omega$, and $1/2$ otherwise. The satisfaction relations $[q \vDash^3 \phi]$ and $[\pi \vDash^3 \gamma]$ for states q and paths $\pi = (q_0, (G_0, \mathbb{l}_0), q_1, \dots)$ is defined as follows:

$$\begin{array}{ll@{\qquad}ll}
[q \vDash^3 \mathtt{tt}] & = 1 & [q \vDash^3 \mathbb{l}] & = L(q, \mathbb{l}) \\
[q \vDash^3 \neg\phi] & = \neg^3([q \vDash^3 \phi]) & [q \vDash^3 \phi_1 \wedge \phi_2] & = \mathit{min}([q \vDash^3 \phi_1], [q \vDash^3 \phi_2]) \\
[q \vDash^3 \exists\gamma] & \multicolumn{3}{l}{= \mathit{max}\,\{[\pi \vDash^3 \gamma] \ : \ \pi(0) = q\}} \\
[\pi \vDash^3 \mathbf{X}_\Omega\, \phi] & \multicolumn{3}{l}{= \mathit{min}([\pi(1) \vDash^3 \phi], D_\Omega(\pi(0), \pi(1)))} \\
[\pi \vDash^3 \phi_1\, \mathbf{U}_\Omega\, \phi_2] & \multicolumn{3}{l}{= \mathit{max}\,\{[\pi \vDash^3 \phi_1\, \mathbf{U}^k_\Omega\, \phi_2] \ : \ k \geq 0\}} \\
[\pi \vDash^3 \phi_1\, \mathbf{U}^k_\Omega\, \phi_2] & \multicolumn{3}{l}{= \mathit{min}(\mathit{min}(\{[\pi(k) \vDash^3 \phi_2]\} \cup \{[\pi(i) \vDash^3 \phi_1] \ : \ i < k\}),} \\
 & \multicolumn{3}{l}{\qquad\quad \mathit{min}\,\{D_\Omega(\pi(i), \pi(i+1)) \ : \ i < k\})}
\end{array}$$

We lift the notion of representation $\triangleright$ from states to paths by defining:

$$\Pi \blacktriangleright \mathsf{E}[\pi] \text{ iff } \forall\, i \geq 0.\ \Pi(i) \triangleright \mathsf{E}[\pi(i)] \wedge \Pi[i] = \pi[i]$$

Furthermore, we define an *information order* $\sqsubseteq$ on truth values by $1/2 \sqsubseteq 0$, $1/2 \sqsubseteq 1$, and $x \sqsubseteq x$ for all $x \in \{0, 1/2, 1\}$. Using this, we can formulate an embedding theorem, which allows us to relate the 2- and 3-valued interpretations of ACTL:

Theorem 1. *Suppose* $(\mathsf{Q}, q_0, \delta, \mathsf{E})$ *faithfully describes the evolution of network* N_0*, and* $\mathcal{T} \vdash N_0 \rightarrow^* N$*. Then:*

1. *If* $N \triangleright \mathsf{E}[q]$ *then* $[q \models^3 \phi] \sqsubseteq [N \models \phi]$.
2. *If* $\Pi \blacktriangleright \mathsf{E}[\pi]$ *then* $[\pi \models^3 \gamma] \sqsubseteq [\Pi \models \gamma]$.

Example 5. For the abstract transition system for $(\mathit{Net}, \mathcal{T}_1)$ of Example 1 and 2, and an Ω containing all possible transition labels, we have

$$[q_0 \models^3 \neg\exists[\mathtt{tt}\ \mathbf{U}_\Omega\ ((\mathtt{l}_1, [\mathtt{has}, \mathtt{l}_2, \mathtt{t}, \mathtt{i}_2]) \wedge (\mathtt{l}_1, [\mathtt{has}, \mathtt{l}_3, \mathtt{t}, \mathtt{i}_3]))]] = 1$$

while on $(\mathit{Net}, \mathcal{T}_2)$ we get the result 1/2. Using Theorem 1, this means that $(\mathit{Net}, \mathcal{T}_1)$ has no evolution such that both $[\mathtt{has}, \mathtt{l}_2, \mathtt{t}, \mathtt{i}_2]$ and $[\mathtt{has}, \mathtt{l}_3, \mathtt{t}, \mathtt{i}_3]$ are exposed tuples at location $\mathtt{l}_1$. In other words, under topology $\mathcal{T}_1$, the node $\mathtt{l}_1$ requesting information on topic $\mathtt{t}$ cannot get replies from both $\mathtt{l}_2$ and $\mathtt{l}_3$. For $(\mathit{Net}, \mathcal{T}_2)$ the analysis states that the abstraction prevents a definite answer.

4 Constructing Abstract Transition Systems

In this section we develop an *algorithm* for constructing an abstract transition system. The development amounts to adapting the approach of [17,18] from the synchronous language CCS to the asynchronous message-passing language bKlaim. This involves solving the challenge of how to deal with the name bindings resulting from message passing. We employ a classical Control Flow Analysis like in [6], using judgements of the form

$$(\hat{\rho}, \hat{S}) \models^G N.$$

The analysis states that $\hat{\rho}$ correctly describes the name bindings and $\hat{S}$ the contents of tuple stores that may take place during the execution of the net N using the connectivity graph G. In the subsequent development we shall not need the $\hat{S}$ component as it will be modelled more precisely using the multisets of exposed labels. We leave the details to the technical report [12].

4.1 Transfer Functions

The abstraction function $\mathcal{E}$ only gives us the information of interest for the initial network. We shall now present auxiliary functions allowing us to approximate how the information evolves during the execution of the network.

Once an action has participated in an interaction, some new actions may become exposed and some may cease to be exposed. We shall now introduce two functions $\mathcal{G}^G_{\hat\rho}$ and $\mathcal{K}$ approximating this information. The relevant information will be an element of:

$$\mathfrak{T} = \mathbf{Loc} \times (\mathbf{Lab} \cup \mathbf{Val}^*) \to \mathfrak{M}$$

As for exposed actions it is not sufficient to use sets: there may be more than one occurrence of an action that is either generated or killed by another action. The ordering $\le_{\mathfrak{T}}$ is defined as the pointwise extension of $\le_{\mathfrak{M}}$.

Generated Actions. To calculate generated actions, we shall introduce the function $\mathcal{G}^G_{\hat\rho} : \mathbf{Net} \to \mathfrak{T}$ which takes a network N and computes an *over*-approximation of which actions might be generated in N:

$$\begin{aligned}
\mathcal{G}^G_{\hat\rho}[\![N_1 \parallel N_2]\!] &= \mathcal{G}^G_{\hat\rho}[\![N_1]\!] \sqcup_{\mathfrak{T}} \mathcal{G}^G_{\hat\rho}[\![N_2]\!] & \mathcal{G}^G_{\hat\rho,l}[\![\mathsf{nil}]\!] &= \bot_{\mathfrak{T}}\\
\mathcal{G}^G_{\hat\rho}[\![l :: P]\!] &= \mathcal{G}^G_{\hat\rho,l}[\![P]\!] & \mathcal{G}^G_{\hat\rho,l}[\![a^\ell.P]\!] &= \tilde{\mathcal{G}}^G_{\hat\rho,l}[\![a^\ell.P]\!] \sqcup_{\mathfrak{T}} \mathcal{G}^G_{\hat\rho,l}[\![P]\!]\\
\mathcal{G}^G_{\hat\rho}[\![l :: S]\!] &= \bot_{\mathfrak{T}} & \mathcal{G}^G_{\hat\rho,l}[\![P_1 \mid P_2]\!] &= \mathcal{G}^G_{\hat\rho,l}[\![P_1]\!] \sqcup_{\mathfrak{T}} \mathcal{G}^G_{\hat\rho,l}[\![P_2]\!]\\
& & \mathcal{G}^G_{\hat\rho,l}[\![A]\!] &= \mathcal{G}^G_{\hat\rho,l}[\![P]\!] \text{ if } A \triangleq P
\end{aligned}$$

$$\begin{aligned}
\tilde{\mathcal{G}}^G_{\hat\rho,l}[\![\mathsf{bcst}^\ell(t).P]\!] &= \bot_{\mathfrak{T}}[(l,\ell) \mapsto \mathcal{E}_l[\![P]\!] +_{\mathfrak{M}} (\textstyle\sum_{\mathfrak{M},l' \in G(l), u \in \hat\rho[\![t]\!]} \bot_{\mathfrak{M}}[(l',u) \mapsto 1])]\\
\tilde{\mathcal{G}}^G_{\hat\rho,l}[\![\mathsf{out}^\ell(t).P]\!] &= \bot_{\mathfrak{T}}[(l,\ell) \mapsto \mathcal{E}_l[\![P]\!] +_{\mathfrak{M}} (\textstyle\sum_{\mathfrak{M},u \in \hat\rho[\![t]\!]} \bot_{\mathfrak{M}}[(l,u) \mapsto 1])]\\
\tilde{\mathcal{G}}^G_{\hat\rho,l}[\![\mathsf{in}^\ell(T).P]\!] &= \bot_{\mathfrak{T}}[(l,\ell) \mapsto \mathcal{E}_l[\![P]\!]]
\end{aligned}$$

Note that the function carries two more parameters, namely a connectivity graph G and the environment $\hat\rho$ which we obtain from the Control Flow Analysis (see above) and which describes the occurring name bindings. The connectivity graph G is needed because it determines at which locations tuples are generated when using broadcast. Likewise, we need $\hat\rho$ to correctly determine which tuples might be output; it is therefore assumed in the following that $(\hat\rho, \hat S) \models^{\bigsqcup \mathcal{T}} N_0$ holds (where $\bigsqcup \mathcal{T}$ is the graph which contains the edges of all $G \in \mathcal{T}$), as it can be shown to imply $(\hat\rho, \hat S) \models^G N_0$ for all $G \in \mathcal{T}$.

All actions $a^\ell.P$ then expose $\mathcal{E}_l[\![P]\!]$, i.e. the actions of the continuation process. Furthermore, $\mathsf{bcst}^\ell(t)$ exposes the tuples $u \in \hat\rho[\![t]\!]$ for all locations $l' \in G(l)$ in the neighbourhood of the sending process; note that $\hat\rho[\![t]\!]$ describes an overapproximation of the ground tuples t can evaluate to. The action $\mathsf{out}^\ell(t)$ exposes all $u \in \hat\rho[\![t]\!]$ only at location l. Analogous to the case for exposed actions, a least fixed point of $\mathcal{G}^G_{\hat\rho}$ can be obtained.

We can show that the the information computed by $\mathcal{G}^G_{\hat\rho}$ is invariant under the structural congruence and potentially *decreases* with network reduction:

Lemma 2. *Suppose* $(\hat\rho, \hat S) \models^{\bigsqcup \mathcal{T}} M$ *holds. If* $M \equiv N$*, then* $\mathcal{G}^G_{\hat\rho}[\![M]\!] = \mathcal{G}^G_{\hat\rho}[\![N]\!]$*. Furthermore, if* $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}}_G N$*, then* $\mathcal{G}^G_{\hat\rho}[\![N]\!] \le_{\mathfrak{T}} \mathcal{G}^G_{\hat\rho}[\![M]\!]$*.*

Note that the function $\mathcal{G}^G_{\hat\rho}$ is defined on pairs of locations and actions only. It can be trivially extended to the general label $\mathbb{l} = (l, \ell[t])$ which is used in the reduction rule for in by defining $\mathcal{G}^G_{\hat\rho}[\![N]\!](l, \ell[t]) = \mathcal{G}^G_{\hat\rho}[\![N]\!](l, \ell)$.

Killed Actions. We define the function $\mathcal{K} : \mathbf{Net} \to \mathfrak{T}$ which takes a network N and computes an *under*-approximation of which actions might be killed in N:

$$\begin{aligned}
\mathcal{K}[\![N_1 \parallel N_2]\!] &= \mathcal{K}[\![N_1]\!] \sqcap_{\mathfrak{T}} \mathcal{K}[\![N_2]\!] & \mathcal{K}_l[\![\mathsf{nil}]\!] &= \top_{\mathfrak{T}} \\
\mathcal{K}[\![l :: P]\!] &= \mathcal{K}_l[\![P]\!] & \mathcal{K}_l[\![a^{\ell}.P]\!] &= \top_{\mathfrak{T}}[(l,\ell) \mapsto \bot_{\mathfrak{M}}[(l,\ell) \mapsto 1]] \sqcap_{\mathfrak{T}} \mathcal{K}_l[\![P]\!] \\
\mathcal{K}[\![l :: S]\!] &= \top_{\mathfrak{T}} & \mathcal{K}_l[\![P_1 \mid P_2]\!] &= \mathcal{K}_l[\![P_1]\!] \sqcap_{\mathfrak{T}} \mathcal{K}_l[\![P_2]\!] \\
& & \mathcal{K}_l[\![A]\!] &= \mathcal{K}_l[\![P]\!] \text{ if } A \triangleq P
\end{aligned}$$

Note that when actions $a^{\ell}.P$ execute at location l, it is clear that one occurrence (l, ℓ) can be killed. A greatest fixed point of $\mathcal{K}$ can be obtained.

We can show that the the information computed by $\mathcal{K}$ is invariant under the structural congruence and potentially *increases* with network reduction:

Lemma 3. *If $M \equiv N$, then $\mathcal{K}[\![M]\!] = \mathcal{K}[\![N]\!]$. Furthermore, if $\mathcal{T} \vdash M \xrightarrow{\mathbb{1}}_G N$ then $\mathcal{K}[\![M]\!] \leq_{\mathfrak{T}} \mathcal{K}[\![N]\!]$.*

Analogously to the case of $\mathcal{G}^G_{\hat{\rho}}$ we can define an extension of $\mathcal{K}$ by

$$\mathcal{K}[\![N]\!](l, \ell[t]) = \mathcal{K}[\![N]\!](l, \ell) +_{\mathfrak{M}} \bot_{\mathfrak{M}} [(l, t) \mapsto 1]$$

i.e. an input action additionally removes a tuple t from the tuple space.

We can use $\mathcal{G}^G_{\hat{\rho}}$ and $\mathcal{K}$ to obtain a transfer function as in a classical Monotone Framework, where E represents a set of exposed actions, and $\mathcal{K}[\![N_0]\!](\mathbb{1})$ (resp. $\mathcal{G}^G_{\hat{\rho}}[\![N_0]\!](\mathbb{1})$) represent the actions which are no longer (resp. newly) exposed by a transition with label $\mathbb{1}$:

$$\mathsf{transfer}_{(G,\mathbb{1}),\hat{\rho}}(E) = (E -_{\mathfrak{M}} \mathcal{K}[\![N_0]\!](\mathbb{1})) +_{\mathfrak{M}} \mathcal{G}^G_{\hat{\rho}}[\![N_0]\!](\mathbb{1})$$

Example 6. Continuing Example 4, we can calculate that

$$\begin{aligned}
\mathcal{K}[\![Net]\!](\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2]) &= [(\mathtt{l}_2, 4) \mapsto 1, (\mathtt{l}_2, [\mathtt{t}, \mathtt{i}_2]) \mapsto 1] \\
\mathcal{G}^G_{\hat{\rho}}[\![Net]\!](\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2]) &= [(\mathtt{l}_2, 5) \mapsto 1]
\end{aligned}$$

and hence that $\mathsf{E}[q_6] = (\mathsf{E}[q_3] -_{\mathfrak{M}} \mathcal{K}[\![Net]\!](\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2])) +_{\mathfrak{M}} \mathcal{G}^G_{\hat{\rho}}[\![Net]\!](\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2])$.

Correctness. The following result states that the transfer function provides safe approximations to the exposed actions of the resulting network:

Theorem 2. *Consider the network* $\mathsf{let}\ A_1 \triangleq P_1; \ldots; A_k \triangleq P_k\ \mathsf{in}\ N_0$ *and suppose* $(\hat{\rho}, \hat{S}) \models^{\bigsqcup \mathcal{T}} N_0$. *If $\mathcal{T} \vdash N_0 \to^* M \xrightarrow{\mathbb{1}}_G N$ then $\mathcal{E}[\![N]\!] \leq_{\mathfrak{M}} \mathsf{transfer}_{(G,\mathbb{1}),\hat{\rho}}(\mathcal{E}[\![M]\!])$.*

4.2 Worklist Algorithm

We are interested in analysing networks N_0 for which we assume in the following that $(\hat{\rho}, \hat{S}) \models^{\bigsqcup \mathcal{T}} N_0$ holds. We shall now construct an abstract transition system which faithfully describes the evolution of N_0 as specified in §3.2.

The key algorithm is a *worklist algorithm*, which starts out from the initial state and constructs the abstract transition system by adding more and more states and transitions. The algorithm makes use of several auxiliary operations:

– Given a state q_s representing some exposed actions, $\mathsf{enabled}$ selects those labels $\mathbb{l}$ that represent actions that may interact in the next step.
– Once the labels $\mathbb{l}$ have been selected, we can use the function $\mathsf{transfer}$ of §4.1.
– Finally, update constructs an appropriate target state q_t and records the transition $(q_s, (G, \mathbb{l}), q_t)$.

The algorithm's main data structures are: A set Q of the current states; a worklist W being a subset of Q and containing those states that have yet to be processed; and, a set δ of the current transitions. The algorithm has the following form:

```
1 $\mathsf{Q} := \{q_0\}$;  $\mathsf{E}[q_0] := \mathcal{E}[\![N_0]\!]$;  $\mathsf{W} := \{q_0\}$;  $\delta := \emptyset$;
2 while $W \neq \emptyset$ do
3     select $q_s$ from $\mathsf{W}$;  $\mathsf{W} := \mathsf{W}\backslash\{q_s\}$;
4     foreach $G \in \mathcal{T}$ do
5         foreach $\mathbb{l} \in \mathsf{enabled}_{(\hat{\rho},\hat{S})}(\mathsf{E}[q_s])$ do
6             let $E = \mathsf{transfer}_{(G,\mathbb{l}),\hat{\rho}}(\mathsf{E}[q_s])$ in $\mathsf{update}(q_s, (G, \mathbb{l}), E)$
```

In line 1 both the set of states and the worklist are initialised to contain the initial state q_0, and q_0 is associated with the set of the exposed actions of the initial network $\mathcal{E}[\![N_0]\!]$. The transition relation δ is empty.

The algorithm then loops over the contents of the worklist W by selecting a q_s it contains, and removing it from W (line 3). For each $G \in \mathcal{T}$ and enabled action $\mathbb{l} \in \mathsf{enabled}_{(\hat{\rho},\hat{S})}(\mathsf{E}[q_s])$ (lines 4–5) the procedure $\mathsf{transfer}_{(G,\mathbb{l}),\hat{\rho}}(\mathsf{E}[q_s])$ returns an extended multiset describing the denotation of the target state. The last step is to update the automaton to include the new transition step, and this is done in line 6 by the procedure call $\mathsf{update}(q_s, (G, \mathbb{l}), E)$.

Procedure update. The procedure update is specified as follows:

```
1 procedure $\mathsf{update}(q_s, (G, \mathbb{l}), E)$
2     if there exists $q \in \mathsf{Q}$ with $H(\mathsf{E}[q]) = H(E)$ then $q_t := q$
3     else select $q_t$ from outside $\mathsf{Q}$;  $\mathsf{Q} := \mathsf{Q} \cup \{q_t\}$;  $\mathsf{E}[q_t] := \bot_{\mathfrak{M}}$;
4     if $\neg(E \leq_{\mathfrak{M}} \mathsf{E}[q_t])$ then $\mathsf{E}[q_t] := \mathsf{E}[q_t] \nabla_{\mathfrak{M}} E$;  $\mathsf{W} := \mathsf{W} \cup \{q_t\}$;
5     $\delta := \delta\backslash\{(q_s, (G, \mathbb{l}), q) \;:\; q \in Q\} \cup \{(q_s, (G, \mathbb{l}), q_t)\}$;
```

First, the target state q_t is determined in lines 2–3, where the reusability of a state is checked by using a *granularity function* H, which is described below.

In line 4 it is ensured that the description $\mathsf{E}[q_t]$ includes the required information E by using a widening operator $\nabla_{\mathfrak{M}}$ in such a way that termination of the overall algorithm is ensured. We shall return to the definition of $\nabla_{\mathfrak{M}}$ below.

The transition relation is updated in line 5. The triple $(q_s, (G, \mathbb{l}), q_t)$ is added, but we also have to remove any previous transitions from q_s with label $(G, \mathbb{l})$, as its target states may be no longer correct. As a consequence, the automaton may contain unreachable parts, which can be removed at this point or after the completion of the algorithm by a simple clean-up procedure for Q, W, and δ.

Granularity Function. The most obvious choice for a granularity function $H : \mathfrak{M} \to \mathcal{H}$ might be the identity function, but it turns out that this choice may lead to nontermination of the algorithm. A more interesting choice is $H(E) = dom(E)$, meaning that only the domain of the extended multiset is of interest; we have used this choice to compute our examples. We can show that termination of the algorithm is ensured once H is chosen to be *finitary*, meaning that $\mathcal{H}$ is finite on all finite sets of labels.

Widening Operator. The widening operator $\nabla_{\mathfrak{M}} : \mathfrak{M} \times \mathfrak{M} \to \mathfrak{M}$ is defined by:

$$(M_1 \nabla_{\mathfrak{M}} M_2)(\mathbb{l}) = \begin{cases} M_1(\mathbb{l}) \text{ if } M_2(\mathbb{l}) \leq M_1(\mathbb{l}) \\ M_2(\mathbb{l}) \text{ if } M_1(\mathbb{l}) = 0 \wedge M_2(\mathbb{l}) > 0 \\ \infty \qquad \text{otherwise} \end{cases}$$

It will ensure that the chain of values taken by $\mathsf{E}[q_t]$ in line 8 always stabilises after a finite number of steps. We refer to [4,14] for a formal definition of widening and merely note that $M_1 \sqcup_{\mathfrak{M}} M_2 \leq_{\mathfrak{M}} M_1 \nabla_{\mathfrak{M}} M_2$.

Procedure enabled. Recall that E is the extended multiset of exposed actions in the state of interest, and remember that $(\hat{\rho}, \hat{S}) \models^{\bigsqcup \mathcal{T}} N_0$ holds. Then:

$$\begin{aligned}\mathsf{enabled}_{(\hat{\rho},\hat{S})}(E) = dom(E) \cap (\ &\{(l, \ell) \ : \ \ell \text{ labels an } \mathsf{bcst}\text{- or } \mathsf{out}\text{-action}\} \cup \\ &\{(l, \ell[t]) \ : \ \ell \text{ labels an } \mathsf{in}(T)\text{-action, } t \in \hat{\rho}[\![T]\!] \text{ and } E(l,t) > 0\}\ )\end{aligned}$$

First of all, $\mathsf{enabled}_{(\hat{\rho},\hat{S})}(E)$ shall only contain labels $\mathbb{l}$ which are exposed in E, hence $\mathbb{l} \in dom(E)$. Furthermore, if ℓ is the label of an bcst- or out-action, then $(l, \ell) \in \mathsf{enabled}_{(\hat{\rho},\hat{S})}(E)$, because these actions can always execute; and if ℓ is the label of an $\mathsf{in}(T)$-action, we have to check which tuples t contained in E match the template T and can be input, and record $(l, \ell[t]) \in \mathsf{enabled}_{(\hat{\rho},\hat{S})}(E)$.

Correctness. We can now establish the main result which implies that we can use the worklist algorithm to produce abstract transition systems for which the embedding theorem (Theorem 1) is applicable.

Theorem 3. *Suppose $(\hat{\rho}, \hat{S}) \models^{\bigsqcup \mathcal{T}} N_0$ holds for a network N_0 and a network topology $\mathcal{T}$, and furthermore that the worklist algorithm terminates and produces an abstract transition system $\mathcal{A}$. Then $\mathcal{A}$ faithfully describes the evolution of N_0.*

5 Conclusion

In this paper, we have dealt with the problem of analysing the behaviour of broadcast networks under changing network connectivity. For networks modelled in the calculus bKlaim, we have defined an algorithm which constructs a finite automaton such that all transition sequences obtained by the evolution of a network correspond to paths in this automaton. We captured the nature of our abstraction by defining a 3-valued interpretation of a temporal logic such that a formula evaluating to a definite truth value on the automaton would imply the truth or falsity of that formula on the transition system of the concrete network.

As a main direction for future work, we would like to construct the abstract transition system as a 3-valued structure itself [8], to model the cases where we can show that progress is enforced.

References

1. Bettini, L., et al.: The Klaim project: theory and practice. In: Priami, C. (ed.) GC 2003. LNCS, vol. 2874, Springer, Heidelberg (2003)
2. Bhargavan, K., Obradovic, D., Gunter, C.A.: Formal verification of standards for distance vector routing protocols. Journal of the ACM 49(4), 538–576 (2002)
3. Bruns, G., Godefroid, P.: Model checking partial state spaces with 3-valued temporal logics. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 274–287. Springer, Heidelberg (1999)
4. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Principles of Programming Languages (POPL'79). Principles of Programming Languages, pp. 269–282. ACM Press, New York (1979)
5. Ene, C., Muntean, T.: A broadcast-based calculus for communicating systems. In: Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'03) (2001)
6. Hansen, R.R., Probst, C.W., Nielson, F.: Sandboxing in myKlaim. In: Availability, Reliability and Security (ARES'06), pp. 174–181. IEEE Computer Society Press, Los Alamitos (2006)
7. Kleene, S.C.: Introduction to Metamathematics. Biblioteca Mathematica, vol. 1. North-Holland, Amsterdam (1952)
8. Larsen, K.G., Thomsen, B.: A modal process logic. In: Logic in Computer Science (LICS'88), pp. 203–210. IEEE Computer Society Press, Los Alamitos (1988)
9. Merro, M.: An observational theory for mobile ad hoc networks. In: Mathematical Foundations of Programming Semantics (MFPS'07). Electronic Notes in Theoretical Computer Science, vol. 173, pp. 275–293 (2007)
10. Nanz, S.: Specification and Security Analysis of Mobile Ad-Hoc Networks. PhD thesis, Imperial College London (2006)
11. Nanz, S., Hankin, C.: A framework for security analysis of mobile wireless networks. Theoretical Computer Science 367(1-2), 203–227 (2006)
12. Nanz, S., Nielson, F., Nielson, H.R.: Topology-dependent abstractions of broadcast networks. Technical report IMM-TR-2007-11, Technical University of Denmark (2007)
13. Nicola, R.D., Vaandrager, F.W.: Action versus state based logics for transition systems. In: Guessarian, I. (ed.) Semantics of Systems of Concurrent Processes. LNCS, vol. 469, pp. 407–419. Springer, Heidelberg (1990)
14. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer, Heidelberg (1999)
15. Nielson, F., Nielson, H.R., Sagiv, M.: A Kleene analysis of mobile ambients. In: Smolka, G. (ed.) ESOP 2000 and ETAPS 2000. LNCS, vol. 1782, pp. 305–319. Springer, Heidelberg (2000)
16. Nielson, F., Nielson, H.R., Sagiv, M.: Kleene's logic with equality. Information Processing Letters 80, 131–137 (2001)
17. Nielson, H.R., Nielson, F.: A monotone framework for CCS. (submitted for publication, 2006)
18. Nielson, H.R., Nielson, F.: Data flow analysis for CCS. In: Program Analysis and Compilation. Theory and Practice. LNCS, vol. 4444, Springer, Heidelberg (2007)
19. Prasad, K.V.S.: A calculus of broadcasting systems. Science of Computer Programming 25(2-3), 285–327 (1995)
20. Sagiv, M., Reps, T., Wilhelm, R.: Parametric shape analysis via 3-valued logic. In: Principles of Programming Languages (POPL'99), pp. 105–118. ACM Press, New York (1999)

On the Expressive Power of Global and Local Priority in Process Calculi

Cristian Versari, Nadia Busi, and Roberto Gorrieri

Università di Bologna, Dipartimento di Scienze dell'Informazione
Mura Anteo Zamboni 7, 40127 Bologna, Italy
{versari,busi,gorrieri}@cs.unibo.it

Abstract. Priority is a frequently used feature of many computational systems. In this paper we study the expressiveness of two process algebras enriched with different priority mechanisms. In particular, we consider a finite (i.e. recursion-free) fragment of asynchronous CCS with global priority (FAP, for short) and Phillips' CPG (CCS with local priority), and we contrast their expressive power with that of two non-prioritised calculi, namely the π-calculus and its broadcast-based version, called bπ. We prove, by means of leader-election-based separation results, that there exists no encoding of FAP into π-Calculus or CPG, under certain conditions. Moreover, we single out another problem in distributed computing, we call the *last man standing* problem (LMS for short), that better reveals the gap between the two prioritised calculi above and the two non prioritised ones, by proving that there exists no parallel-preserving encoding of the prioritised calculi into the non-prioritised calculi retaining any *sincere* (complete but partially correct, i.e., admitting divergence or premature termination) semantics.

1 Introduction

Priority is a frequently used feature of many computational systems. High-priority processes dispose of more central processing unit time in workstations, or preempt the execution of low priority processes through hardware/software-driven interrupt mechanisms. In order to model such systems, many basic process algebras have been enriched with some priority mechanisms (see, e.g., [1,2,3,4,5]). Priority is also implicitly used in many stochastic process calculi, where immediate actions take precedence over timed actions (see, e.g., [6,7,8], or where actions are equipped with an explicit priority level (e.g., [9]).

In this paper we investigate the expressiveness of priority in (untimed) concurrent systems, in order to delineate the expressive power gained by the addition of priority and to compare the relative expressive power of different priority mechanisms, by studying a couple of problems in distributed systems.

According to the classification in [4], the basic priority mechanisms reported in the literature can be divided into two main groups: those based on *global* priority (see, e.g., [10,3]) and those based on *local* priority (see, e.g., [2,5]). The difference is motivated by the *scope* of the priority effects on the system: in the case of global priority, a high-priority action is able to preempt any other low-priority action in the system, so that only higher priority processes are allowed to evolve. In the case of local priority, this effect is limited to the *location* of the process, where a location can be thought as a site on a distributed system and may be represented by the scope of some name or the guarded choice between actions with different priorities. An example may be helpful in showing the difference between the two. Consider the system S composed of five processes

$$S \equiv \overline{a}.P \;\big|\; a.Q_1 + \underline{b}.Q_2 \;\big|\; \overline{\underline{b}}.R \;\big|\; c.T_1 + d.T_2 \;\big|\; \overline{c}.V$$

where the sum operator represents the usual choice between different actions, output actions are overlined and high-priority actions are underlined. According to the semantics of CCS^{sg} (CCS with a global notion of priority) and CCS^{sl} (CCS with local priority) reported in [4], the processes $a.Q_1 + \underline{b}.Q_2$ and $\overline{\underline{b}}.R$ are ready to perform a high-priority action on channel $\underline{b}$. In CCS^{sg} semantics this action is forced to happen before any other low priority transition in S, while in CCS^{sl} semantics, the action $\underline{b}$ only preempts the execution of the action a, so that the synchronisation on c of the last two processes may even happen first. In other words, with a global priority notion the only possible internal transition of the system S is

$$S \rightarrow \overline{a}.P \;\big|\; Q_2 \;\big|\; R \;\big|\; c.T_1 + d.T_2 \;\big|\; \overline{c}.V$$

while in presence of local priority also the evolution

$$S \rightarrow \overline{a}.P \;\big|\; a.Q_1 + \underline{b}.Q_2 \;\big|\; \overline{\underline{b}}.R \;\big|\; T_1 \;\big|\; V$$

can happen. In both cases only the reaction on channel a is preempted.

As a basic representative for a calculus with global priority, we consider in this paper a very minimal fragment, we call FAP, of CCS [11] (without restriction and recursion, with asynchronous communication), enriched with static priority and global preemption (like in CCS^{sg} reported in [4]) where only the prefix operator on inputs is present and the asynchronous output is characterised by the possibility of assigning different priority to the outgoing messages.

As a representative for a calculus with local priority, we consider in this paper Phillips' CCS with priority guards (CPG for short) [5].

Moreover, we consider two well-known unprioritised calculi, namely the π-calculus [12,13] and its broadcast-based version bπ-Calculus [14], that will be compared with the two prioritised calculi above.

The two problems in distributed systems we will use to distinguish the expressive power of these four calculi are the *leader-election* problem [15], already used to study expressiveness gap between, e.g., synchronous and asynchronous π-calculus [16], and an apparently new problem we have called *the last man standing* problem (LMS for short), consisting in the capability for processes of recognising the absence of other processes ready to perform synchronisations or input/output operations. In other words, the LMS problem is solvable if a process is able to check that it is the only one active in a network.

1.1 Contribution of This Paper

The first application of the leader election problem to the π-Calculus [16] allowed to observe the superior expressiveness of mixed-choice with respect to separated-choice in the π-Calculus. Following the same approach many other separation results have been obtained, each one based on the capability or impossibility to solve the leader election under appropriate conditions. An adapted version was used in [14] to show that the broadcast-based bπ-Calculus, is more expressive than the standard π-Calculus provided with point-to-point communication primitives. The result is based on the idea that broadcast permits to solve the leader election problem even without any knowledge of the number of processes participating to the election.

The same approach was used in [5] to separate CPG from CCS and also from the π-Calculus by exploiting the broadcast-like power of preemption introduced by priority, while π-Calculus capability of creating new communication links between processes is exploited to prove the converse separation from CPG. This result is the only on the expressiveness of priorities in process algebras we are aware of.

In this paper we first analyse the expressiveness of global priority, in order to check if it can be proved to be more expressive than local priority as it would be natural expecting. In order to represent a global priority model we choose FAP, a fragment of CCS added with stratified, global priority (a slight variant of the CCS with static priority and global preemption, CCS^{sg} studied in [4]) where the only operator is the prefix on inputs, while the asynchronous output models the dispatch of messages with two different levels of priority: the delivery of high-priority messages is ensured to happen before that of low-priority ones.

We prove that this very simple language (deprived of synchronous communication, choice, recursion or replication and hence finite) consents to write programs capable of solving the leader election problem in any connected graph of processes without knowledge of the number of processes involved in the election.

By applying the idea used in [14,5] we have as corollary that FAP cannot be distributively encoded into the π-Calculus, but we prove this to remain true also for partially correct encodings which introduce divergence or failure in their computations as consequences of livelocked or deadlocked conditions. This result can also be extended to the translations of bπ-Calculus and CPG into the π-Calculus, thus relaxing the encoding conditions already stated in [14,5].

Another consequence of the above leader election result in FAP is the impossibility of its encoding into CPG under uniformity and independence-preserving conditions, which constitutes the expected result on the expressiveness gap between global and local priority in the chosen process algebra framework. It is worth considering that the separation between these two prioritised languages and the π-Calculus is stronger than that between FAP and CPG themselves, which is a first hint on the expressive power of priority on both global and local approaches.

In order to strengthen the separation between prioritised and non-prioritised languages, we then introduce a new setting denoted as the *last man standing*

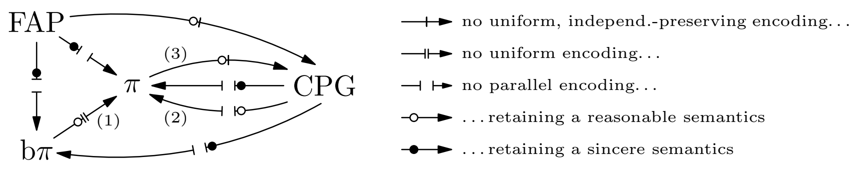

FAP
π
CPG
bπ
(1)
(2)
(3)
no uniform, independ.-preserving encoding. . .
no uniform encoding. . .
no parallel encoding. . .
. . . retaining a reasonable semantics
. . . retaining a sincere semantics

Fig. 1. Impossibility results: (1) C.Ene, T.Muntean [14]; (2, 3) I. Phillips [5,17]; the remaining ones are presented in this paper

problem. In this setting a bunch of n processes must realise if there is only one process participating to the LMS (and in that case, the only process would be the "last man standing"), i.e. understand if $n = 1$ or $n > 1$ in a distributed way. We prove that it is possible to solve the LMS both in FAP and CPG (but we claim that it is also possible within other priority approaches like [2,4]), while it is not possible in non-prioritised languages like the π-Calculus but also the bπ-Calculus. This result implies that there exist no distributed encodings of FAP and CPG into the bπ-Calculus, hence showing that the greatest expressiveness of priority does not derive from the broadcast-like power of preemption, but from the capability of processes to know if another process is ready to perform a synchronisation on some channel *or not*. In non-prioritised calculi it is possible to know if some process *is* ready to perform some synchronisation, but it is not decidable if, on the contrary, the condition does not hold. We show that in a distributed setting this simple capability is proper of priority (global or local, stratified or not) and cannot be obtained anyhow even if providing broadcast-like primitives and admitting divergent or deadlocked computations.

The above results are summarised in figure 1.

1.2 Structure of the Paper

The rest of the paper is structured as follows. In section 2 the four process algebras involved in the separation results are introduced and a brief explanation of their main features is given. Section 3 contains a short discussion and the formal definitions of the properties of an encoding. In Sect. 3.1 and 3.2 the leader election and LMS problems are formalised. In Sect. 4.1 and 4.2 the respective separation results are shown, then some conclusive remarks are reported.

2 Calculi

We introduce now the calculi of interest in this paper by giving their syntax and a short explanation of their functioning while for the respective semantics we refer to [18,19,14,5,17] for lack of space, except for FAP whose definition is comprehensive.

2.1 The π-Calculus

The π-Calculus [12,13] is a derivation of CCS [11] where processes interact through synchronisation over named channels, with the capability of receiving new channels and subsequently using them for the interaction with other processes in order to model mobility.

Definition 1. *Let $\mathcal{N}$ be a set of names on a finite alphabet, $x, y, \ldots \in \mathcal{N}$. The syntax of the π-Calculus is defined as*

$$\begin{array}{lcl} P & ::= & \mathbf{0} \;\Big|\; \sum_{i\in I} \pi_i.P_i \;\Big|\; P \mid Q \;\Big|\; !P \;\Big|\; (\nu x)P \\ \pi & ::= & \tau \;\Big|\; x(y) \;\Big|\; \overline{x}\langle y\rangle \end{array}$$

where: $\mathbf{0}$ represents the null process, $x(y)$ expresses the capability of performing an input on the channel x and receiving a datum which is then bound to the name y; $\overline{x}\langle y\rangle$ expresses the capability of sending the name y on the channel x; τ is the invisible, uncontrollable action; $P \mid Q$ represents the parallel composition of processes; $!P$ stands for the unlimited replication of process P; $\sum_{i\in I} \pi_i.P_i$ represents the nondeterministic choice between several input / output communication capabilities, denoted also as $\pi_1.P_1 + \pi_2.P_2 + \ldots$; $(\nu x)P$ represents the scope restriction of the name x to process P.

In order to define the observables of a π-Calculus process P, we introduce the notion of barb.

Definition 2. *Let P be a π-Calculus process. P exhibits barb α, written $P \downarrow \alpha$, iff*

- $P \equiv (\nu\tilde{y})(x(z).Q + R \mid S)$, *with* $\alpha = x$, $x \notin \tilde{y}$ *or*
- $P \equiv (\nu\tilde{y})(\overline{x}\langle z\rangle.Q + R \mid S)$, *with* $\alpha = \overline{x}$, $x \notin \tilde{y}$.

Each barb α represents one action that P is immediately ready to perform.

For a full treatment we refer to [18,19].

2.2 The bπ-Calculus

The bπ-Calculus [14] is a variant of the π-Calculus where the point-to-point synchronisation mechanism is replaced by broadcast communication. For example, while the π-Calculus program

$$S \equiv \overline{a}\langle b\rangle.P \mid a(x).Q \mid a(y).R \mid a(z).T$$

can evolve in one step to a system like S_1

$$S \rightarrow S_1 \equiv P \mid Q\{b/x\} \mid a(y).R \mid a(z).T$$

where only one of Q, R, S is affected by the performed communication, in bπ-Calculus the system S directly evolves to S_2

$$S \rightarrow S_2 \equiv P \mid Q\{b/x\} \mid R\{b/y\} \mid T\{b/z\}$$

where all the processes listening on channel a receive the broadcasted message.

Definition 3. *Let $\mathcal{N}$ be a set of names on a finite alphabet, $x, y, \ldots \in \mathcal{N}$. The syntax of the b$\pi$-Calculus is defined in terms of the following grammar* [1]

$$P \quad ::= \quad \mathbf{0} \;\Big|\; A\langle\tilde{x}\rangle \;\Big|\; \sum\nolimits_{i\in I} \alpha_i.P_i \;\Big|\; P_1 \mid P_2 \;\Big|\; \nu x P \;\Big|\; (\mathrm{rec}\, A\langle\tilde{x}\rangle.P)\langle\tilde{y}\rangle$$

where

$$\alpha_i \quad ::= \quad x(y) \;\Big|\; \overline{x}y \;\Big|\; \nu y\overline{x}y \;\Big|\; \tau$$

As for the π-Calculus, we define the notion of barb in order to express the observable actions a bπ process is ready to perform.

Definition 4. *Let P be a bπ-Calculus process. P exhibits barb α, written $P \downarrow \alpha$, iff one of*

- $P \equiv \sum_{i\in I} \pi_i.P_i$, $\pi_i = x(y)$ *for some i and* $\alpha = x$*;*
- $P \equiv \sum_{i\in I} \pi_i.P_i$, $\pi_i = \overline{x}\langle y\rangle$ *for some i and* $\alpha = \overline{x}$*;*
- $P \equiv P_1 \mid P_2$ *and* $P_1 \downarrow \alpha \vee P_2 \downarrow \alpha$*;*
- $P \equiv \nu a P'$ *and* $P' \downarrow \alpha$*, with* $\alpha, \overline{\alpha} \neq a$*;*
- $P \equiv (recA\langle\tilde{x}\rangle.P')\langle\tilde{y}\rangle$ *and* $P'[(recA\langle\tilde{x}\rangle.P')/A, \tilde{y}/\tilde{x}] \downarrow \alpha$*;*

is satisfied.

The original semantics is given in terms of a labelled transition systems: in order to simplify and uniform the notation, from now on we say that $P \to Q \iff P \xrightarrow{\overline{x}y} Q \vee P \xrightarrow{\nu y\overline{x}y} Q \vee P \xrightarrow{\tau} Q$. For a detailed description of bπ-Calculus syntax and its semantics we refer to [14] for lack of space.

2.3 The CPG Language

The CPG language [5] derives from CCS with the addition of a local notion of priority over actions. Each action is guarded by a set of names representing the actions whose availability may prevent its execution. For example, the system S

$$S \equiv a.P \mid \overline{a}.Q \mid (a : b.R_1 + c : d.R_2) \mid \overline{b}.T \mid \overline{d}.V$$

may perform a synchronisation on channel a, but also on channel d because no action $\overline{c}$ is available. Instead a communication on b is not possible because it is prevented by the presence of the action $\overline{a}$.

Definition 5. *Let $\mathcal{N}$ be a set of names on a finite alphabet, $x, y, \ldots \in \mathcal{N}$. CPG syntax is defined in terms of the following grammar*

$$\begin{aligned} P \quad ::= \quad & \mathbf{0} \;\Big|\; A\langle\tilde{x}\rangle \;\Big|\; \sum\nolimits_{i\in I} S_i : \alpha_i.P_i \;\Big|\; P_1 \mid P_2 \;\Big|\; \\ & \nu x P \;\Big|\; A(x_1, \ldots, x_n) \overset{def}{=} P \end{aligned}$$

where

$$\alpha_i \quad ::= \quad x \;\Big|\; \overline{x} \;\Big|\; \tau$$

and $S_i \subseteq \mathcal{N}$ represents the set of actions whose co-actions availability prevents the execution of α_i.

[1] For sake of clarity we maintain bπ original syntax since for its semantics we refer entirely to [14].

We report the definition of barb for CPG in [17].

Definition 6. *Let P be a CPG process. P exhibits barb α, written $P \downarrow \alpha$, iff*

- $P \equiv (\nu\tilde{y})(S : x.Q + R \mid T)$, *with* $\alpha = x$, $x \notin \tilde{y}$ *or*
- $P \equiv (\nu\tilde{y})(S : \overline{x}.Q + R \mid T)$, *with* $\alpha = \overline{x}$, $x \notin \tilde{y}$.

For a full treatment of the CPG language we refer to [5,17].

2.4 The FAP Language

As previously outlined, the FAP language is a slight variant of a minimal CCS^{sg} fragment, that is CCS added with static, global priority [4]: by keeping FAP minimal the expressive power of global priority can be better isolated. Only two operators are present in FAP: parallel composition and prefix. The prefix operation is allowed only after input actions, so that the output can be considered asynchronous as for the asynchronous π-Calculus (see e.g. [16]). Output actions are characterised by two priority levels, meaning that high priority output synchronisations are guaranteed to happen before low priority ones. As an example, consider the system

$$S \equiv a.P \mid a.Q \mid b.R \mid \overline{a} \mid \underline{\overline{a}} \mid \overline{b}$$

The processes $\overline{a}, \underline{\overline{a}}, \overline{b}$ model messages which must be delivered to the processes listening on the appropriate channels: The message $\underline{\overline{a}}$ has higher priority w.r.t. any other message in S and hence must be delivered before. Consequently the only possible transitions of S are

$$S \rightarrow P \mid a.Q \mid b.R \mid \overline{a} \mid \overline{b} \qquad\qquad S \rightarrow a.P \mid Q \mid b.R \mid \overline{a} \mid \overline{b}$$

where the process receiving the message is nondeterministically chosen. After this transition, either $\overline{a}$ or $\overline{b}$ can finally be delivered. In order to simplify the notation, inputs lack any denotation of priority, but the results presented in this paper are completely independent of this design choice.

Definition 7. *Let $\mathcal{N}$ be a set of names on a finite alphabet, $x, \ldots \in \mathcal{N}$. FAP syntax is defined in terms of the following grammar*

$$P \quad ::= \quad \mathbf{0} \;\Big|\; x.P \;\Big|\; \overline{x} \;\Big|\; \underline{\overline{x}} \;\Big|\; P \mid Q$$

In order to keep it simple as well, we define FAP semantics in terms of a reduction system in the style of [18].

Definition 8. *Structural congruence for FAP is the congruence $\equiv$ generated by the following equations:*

$$P \mid \mathbf{0} \equiv P, \qquad P \mid Q \equiv Q \mid P, \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$

Definition 9. *FAP operational semantics is given in terms of the reduction system described by the following rules:*

$$\frac{}{x.P \mid \overline{x} \mapsto P} \qquad \frac{}{x.P \mid \underline{\overline{x}} \twoheadrightarrow P}$$

$$\frac{P \twoheadrightarrow P'}{P \mid Q \twoheadrightarrow P' \mid Q} \qquad \frac{P \mapsto P' \qquad P \mid Q \not\twoheadrightarrow R}{P \mid Q \mapsto P' \mid Q}$$

$$\frac{P \equiv Q \quad P \mapsto P' \quad P' \equiv Q'}{Q \mapsto Q'} \qquad \frac{P \equiv Q \quad P \twoheadrightarrow P' \quad P' \equiv Q'}{Q \twoheadrightarrow Q'}$$

We say that $P \rightarrow Q \iff P \mapsto Q \vee P \twoheadrightarrow Q$.

Definition 10. *For any process in FAP, the function* fn *is defined as*

$$\mathrm{fn}(\mathbf{0}) = \emptyset \qquad \mathrm{fn}(\overline{x}) = \{x\} \qquad \mathrm{fn}(x.P) = \{x\} \cup \mathrm{fn}(P)$$
$$\mathrm{fn}(\underline{\overline{x}}) = \{x\} \qquad \mathrm{fn}(P \mid Q) = \mathrm{fn}(P) \cup \mathrm{fn}(Q)$$

As for previous languages, we define the notion of barb.

Definition 11. *A FAP process P exhibits barb α, written as $P \downarrow \alpha$, iff*

- $P \equiv x.Q \mid R$, $\alpha = x$, *or*
- $P \equiv \overline{x} \mid R$, $\alpha = \overline{x}$, *or*
- $P \equiv \underline{\overline{x}} \mid R$, $\alpha = \underline{\overline{x}}$.

Proposition 1. *FAP is not Turing-complete.*

Proof. Every process $P \in$ FAP terminates — FAP lacks any loop operator such as bang or recursion.

3 Encodings

In order to provide the results previously outlined, we now formalise the encoding conditions relevant for the expressiveness separation between languages.

We first formalise the notion of *observables* of a program computation, in the style of [17].

Definition 12. *Let L be a process language and processes $P, P_0, \ldots \in L$. A* computation $\mathcal{C}$ of P *is a finite or infinite sequence $P = P_0 \rightarrow P_1 \rightarrow \cdots$. $\mathcal{C}$ is* maximal *if it cannot be extended.*

A computation of a process P is the sequence of states P can reach during its execution. Each process P may present many different computations due to the nondeterminism intrinsic in concurrent calculi.

Definition 13. *Let L be a process language with names in $\mathcal{N}$ and processes $P_0, \ldots, P_i \in L$. Let $\mathcal{C}$ be a computation $P_0 \rightarrow \cdots \rightarrow P_i \cdots$. Given a set of* intended observables $\mathrm{Obs} \subseteq \mathcal{N}$, *the observables of $\mathcal{C}$ are* $\mathrm{Obs}(\mathcal{C}) = \{x \in \mathrm{Obs} : \exists i\, P_i \downarrow x\}$.

The observables of a computation C are the set of all the external interactions the process may perform in the states reached during the computation.

Some of the separation results are based on the topology of the network of processes: for example, the encoding impossibility of the π-Calculus into value-passing CCS [16] is based on the hypothesis that the encoding does not increase the connection of the network, that is all the processes independent (not sharing free names) in the source language must remain independent after the encoding. The same criterion will be necessary to separate FAP and CPG.

Definition 14. *Let L be a process language. Two processes $P, Q \in L$ are* independent *if they do not share any free names, that is* $\mathrm{fn}(P) \cap \mathrm{fn}(Q) = \emptyset$.

We now define the conditions an encoding may preserve, in the style of [17].

Definition 15. *Let L, L' be process languages. An encoding $[\![\cdot]\!] : L \to L'$ is*

1. observation-respecting *if $\forall P \in L$,*
   - *for every maximal computation $\mathcal{C}$ of P there exists a maximal computation $\mathcal{C}'$ of $[\![P]\!]$ such that* $\mathrm{Obs}(\mathcal{C}) = \mathrm{Obs}(\mathcal{C}')$*;*
   - *for every maximal computation $\mathcal{C}$ of $[\![P]\!]$ there exists a maximal computation $\mathcal{C}'$ of P such that* $\mathrm{Obs}(\mathcal{C}) = \mathrm{Obs}(\mathcal{C}')$*;*
2. weakly-observation-respecting *if $\forall P \in L$,*
   - *for every maximal computation $\mathcal{C}$ of P there exists a maximal computation $\mathcal{C}'$ of $[\![P]\!]$ such that* $\mathrm{Obs}(\mathcal{C}) = \mathrm{Obs}(\mathcal{C}')$*;*
   - *for every maximal computation $\mathcal{C}$ of $[\![P]\!]$ there exists a maximal computation $\mathcal{C}'$ of P such that* $\mathrm{Obs}(\mathcal{C}) \subseteq \mathrm{Obs}(\mathcal{C}')$*;*
3. distribution-preserving *if $\forall P_1, P_2 \in L$, $\quad [\![P_1 \mid P_2]\!] = [\![P_1]\!] \mid [\![P_2]\!]$;*
4. renaming-preserving *if for any permutation σ of the source names in L there exists a permutation θ in L' such that $[\![\sigma(P)]\!] = \theta([\![P]\!])$ and the permutations are compatible on observables, that is $\sigma|_{\mathrm{Obs}} = \theta|_{\mathrm{Obs}}$;*
5. independence-preserving *if $\forall P, Q \in L$, if P and Q are independent then $[\![P]\!]$ and $[\![Q]\!]$ are also independent.*

The observation-respecting property is the minimal requirement that any reasonable encoding should preserve: it ensures that some intended observable behaviour is preserved by the translation. The weak variant of this condition admits encodings which introduce divergence or failure deriving from livelocks or deadlocks. Under certain conditions, like fairness or other hypotheses on scheduling or execution, the introduction of divergence or failure by the encoding may be tolerated [20,21] because it would be guaranteed not (or very unlikely) to happen anyway.

The distribution-preserving is a very important feature of an encoding in a concurrent framework: it implies its compositionality and above all it guarantees that the degree of parallelism of the translated system does not decrease.

The renaming-preserving property states that the encoding should not introduce asymmetries in the system, essential condition to preserve when impossibility results on problems such as the leader election are completely based on the symmetric topology of the network.

Independence-preserving represents the property of not increasing the connection of the network.

According to [16,17], a distribution- and renaming-preserving encoding is called *uniform*, and *reasonable* if it preserves the intended observables over maximal computations. We call *sincere* a weakly-observation-respecting encoding, which is complete but only weakly correct, in the meaning that it admits divergence or premature termination.

3.1 The Leader Election Problem

An electoral system represents the situation of a bunch of processes (modelling for example a set of workstations in a network) aiming at reaching a common agreement, i.e. deciding which one of them (an no other) is the leader. The modelling of the election problem in process algebras requires that the system composed of the network of processes will sooner or later signal unequivocally the result of the election on some channels ω_i, which become consequently the observables of interest on the computations of the system.

Definition 16. *Let L be a process language, and processes $P_1, \ldots, P_k \in L$. A network* $\mathsf{N}et$ *of size k, with $k \geq 1$, is a system* $\mathsf{N}et = P_1 \mid \ldots \mid P_k$.

Definition 17. *A network* $\mathsf{N}et$ *of size k is an* electoral system *if for every maximal computation $\mathcal{C}$ of* $\mathsf{N}et$ $\exists i \leq k : \mathrm{Obs}(\mathcal{C}) = \{\omega_i\}$, *where* $\mathrm{Obs} = \{\omega_i : i \in \mathbb{N}\}$.

In order to keep notation simple, the definition of electoral system reflects the design choices kept in [16,14,5] which are based on the hypothesis that the system will never perform external interactions on channel which are not intended to be observable. As in [5,17], the winner process (the leader) is supposed to signal the outcome of the election, while all the other processes simply do nothing.

Some additional definitions are given in order to formalise the idea of connected network, in the style of [16].

Definition 18. *A* hypergraph *of size k is a tuple $H = \langle N, X, t \rangle$ where N is a finite set of* nodes, $|N| = k$, *X a finite set of* edges, *and $t : X \to 2^N$ is a function which assigns to each $x \in X$ a set of nodes, $t(x) = \{n_1, \ldots, n_i\}$. Two nodes $n_1, n_2 \in N$ are neighbours if $\exists x \in X : \{n_1, n_2\} \subseteq t(x)$.*

Definition 19. *Given a network* $\mathsf{N}et = P_1 \mid \ldots \mid P_k$, *the* hypergraph associated *to* $\mathsf{N}et$ *is* $H(\mathsf{N}et) = \langle N, X, t \rangle$ *with* $N = \{1, \ldots, k\}$, $X = \mathrm{fn}(P_1 \mid \ldots \mid P_k)$, *and* $\forall x \in X, t(x) = \{n : x \in \mathrm{fn}(P_n)\}$.

Definition 20. *A hypergraph $H = \langle N, X, t \rangle$ is* connected *if $|N| = 1$ or $\exists n \in N, x \in X : \{n\} \subset t(x)$ and $H' = \langle N', X, t' \rangle$ is connected, where $N' = N \setminus \{n\}$ and $t'(x) = t(x) \setminus \{n\} \forall x$. H is fully connected if $|N| = 1$ or $\forall n_1, n_2 \in N \ \exists x \in X : \{n_1, n_2\} \subset t(x)$.*

In a connected hypergraph each pair of nodes is connected by a sequence of hyperedges: Def. 20 concisely represents this condition. Nodes are directly connected by an edge in a fully connected hypergraph. We say that a network is (fully) connected if the associated hypergraph is (fully) connected.

3.2 The Last Man Standing Problem

The last man standing represents a very simple situation where a bunch of n processes in a fully connected network must realise if $n = 1$ or $n > 1$ in a distributed way. The possibility or impossibility to solve the LMS problem is based on the idea that in a given language L a process P *may* or *may not* know if another process Q is ready to perform some intended action on a given channel. Usually the only way P has to know the presence of Q is to *try* a synchronisation on it. Since the input (and often also the output) is blocking, P results blocked if the condition does not hold, or it follows another computation without knowledge on the presence of Q. The definition of LMS system follows.

Definition 21. *A network* $\mathsf{N}et_k$ *of size* k *is a* LMS system *if for every maximal computation* $\mathcal{C}$ *of* $\mathsf{N}et_k$

- $\text{Obs}(\mathcal{C}) = \{y\}$ *if* $k = 1$,
- $\text{Obs}(\mathcal{C}) = \{n\}$ *if* $k > 1$,

where $\text{Obs} = \{y, n\}$.

The above definition implies that any LMS system is a fully connected network. It is possible to adapt the definition for not fully connected networks, in order to better exploit the scoping of local priorities and still the separation results based on LMS would hold.

4 Separation Results

The separation results previously outlined follow. We first give those based on the leader election, and then those based on the LMS problem.

4.1 Leader-Election-Based Separation Results

The bπ-Calculus and CPG were proved capable of solving the leader election in a fully connected network without knowledge of the number of processes. Here we show that in FAP this is possible in any (not only fully) connected network.

Theorem 1. *Let* $P_1, \ldots, P_k$ *be FAP processes,* $\mathsf{N}et = P_1 \mid \cdots \mid P_k$ *and* $H = \langle N, X, t\rangle$ *be the hypergraph of size* k *associated to* $\mathsf{N}et$. *Let*

$$\begin{aligned} P_n = \overline{m}_n \mid \underline{s}_n \mid\; & m_n.s_n.(\omega_n \mid \overline{\underline{d}}_{n1} \mid \cdots \mid \overline{\underline{d}}_{nz_n}) \\ \mid\; & d_{n1}.(s_n \mid \overline{\underline{d}}_{n1} \mid \cdots \mid \overline{\underline{d}}_{nz_n}) \\ & \vdots \\ \mid\; & d_{nz_n}.(s_n \mid \overline{\underline{d}}_{n1} \mid \cdots \mid \overline{\underline{d}}_{nz_n}) \end{aligned}$$

where P_i *and* P_h *are neighbours iff* $\exists j, l : d_{ij} = d_{hl}$, *with* ω_i, s_j, m_h *distinct and* $\omega_i \neq s_j \neq m_h \neq d_{pq}, \forall i, j, h, p, q$. *If* H *is connected then* $\mathsf{N}et$ *is an electoral system.*

The following lemma [17] is needed to prove that the above results cannot be obtained without knowledge of the number of processes in the electoral system and also to show that the LMS problem is undecidable in the π-Calculus. As noted in [17], it would also hold for any language having comparable semantics of the parallel operator, such as Mobile Ambients [22].

Lemma 1. *1. For any π-Calculus processes P_1, P_2, if P_i has a maximal computation with observables $O_i (i = 1, 2)$ then $P_1 \mid P_2$ has a maximal computation with observables O such that $O_1 \cup O_2 \subseteq O$.*

Next we state a similar but weaker result for the bπ-Calculus, which is also needed for the separation from FAP and CPG based on the LMS problem.

Lemma 2. *1. For any bπ-Calculus processes P_1, P_2, if P_i has a maximal computation with observables $O_i (i = 1, 2)$ then $P_1 \mid P_2$ has two maximal computations $\mathcal{C}_1, \mathcal{C}_2$ (not necessarily distinct) with respective observables O'_1, O'_2 such that $O_i \subseteq O'_i$.*

The next theorem follows the idea in [14,5]. As discussed for the definition of sincere semantics, here the condition on the preserved observables is weaker than those considered in [14,5] but it is possible to relax them as well.

Theorem 2. *There is no distribution-preserving and weakly-observation-respecting encoding of FAP into the π-Calculus.*

Proof. Consider $P_n = \overline{m}_n \mid \underline{s}_n \mid m_n.s_n.(\omega_n \mid \underline{d}) \mid d.(s_n \mid \underline{d})$ for $n = 1, 2$. By theorem 1, $\mathsf{N}et_1 = P_1$, $\mathsf{N}et_2 = P_2$ and $\mathsf{N}et_{12} = P_1 \mid P_2$ are electoral systems. Let $[\![\cdot]\!]$ be a weakly-observation-respecting and distribution-preserving encoding of FAP into the π-Calculus. Hence $[\![P_1]\!]$ has a maximal computation $\mathcal{C}_1$ with observables $\mathrm{Obs}(\mathcal{C}_1) = \omega_1$, because of the weakly-observation-respecting condition. But also $[\![P_2]\!]$ has a maximal computation $\mathcal{C}_2$ with observables $\mathrm{Obs}(\mathcal{C}_2) = \omega_2$.

So we have, for the distribution-preserving property,

$$[\![\mathsf{N}et_{12}]\!] = [\![P_1 \mid P_2]\!] = [\![P_1]\!] \mid [\![P_2]\!]$$

By lemma 1, $[\![\mathsf{N}et_{12}]\!]$ has a maximal computation $\mathcal{C}_{12}$ with observables $\{\omega_1, \omega_2\} \subseteq \mathrm{Obs}(\mathcal{C}_{12})$, not included in the set of observables of any maximal computation of $\mathsf{N}et_{12}$, which contradicts $[\![\cdot]\!]$ being weakly-observation-preserving.

Theorem 3. *There is no uniform, observation-respecting and independence-preserving encoding of FAP into CPG.*

Proof. (sketch) By theorem 1, it is possible to solve in FAP the leader election in any symmetric ring. The impossibility result is then the same proved for the encoding of the π-Calculus into CPG [17]. It is worth remarking that while the separation between π-Calculus and CPG derives from the capability of communicating new names proper of the π-Calculus, the separation between FAP and CPG is a strict consequence of the different scope of priority in the two languages.

4.2 LMS-Based Separation Results

In this section the separation results based on the last man standing problem are formalised. First we show that both in FAP and CPG the LMS can be solved, and then from this expressive capability we derive the impossibility of encoding FAP or CPG into π-Calculus or bπ under distribution and weak preservation of observables hypotheses.

Lemma 3. *Let P be a FAP process,*

$$P = \overline{m} \mid \underline{\overline{s}} \mid \underline{\overline{q}} \mid k.(s \mid q \mid \underline{\overline{k}}) \mid m.s.(s.q.(\underline{\overline{k}} \mid n) \mid \overline{l} \mid l.q.y)$$

Then $\mathsf{N}et_k = \underbrace{P \mid \cdots \mid P}_{k}$ *is a LMS system of size k, $\forall k \geq 1$.*

Lemma 4. *Let P be a CPG process,*

$$P = a : b.\overline{a} \mid \overline{b}.(b : \tau.y \mid z : b.(\overline{b} \mid n \mid \overline{z}))$$

Then $\mathsf{N}et_k = \underbrace{P \mid \cdots \mid P}_{k}$ *is a LMS system of size k, $\forall k \geq 1$.*

The following is an alternative way w.r.t. theorem 2 to prove the separation between FAP and π-Calculus and act as a template for the next theorems.

Theorem 4. *There is no distribution-preserving and weakly-observation-respecting encoding $[\![\cdot]\!]$ of FAP into the π-Calculus.*

Proof. Suppose $[\![\cdot]\!]$ is distribution-preserving and weakly-observation-respecting. By lemma 3 $\exists P : \mathsf{N}et_k$ is a LMS system for any $k \geq 1$, where $\mathsf{N}et_k = P \mid \ldots \mid P$. By the weakly-observation-respecting condition, $[\![\mathsf{N}et_1]\!]$ has a computation $\mathcal{C}_1$ with observables $\mathrm{Obs}(\mathcal{C}_1) = \{y\}$. By the distribution-preserving condition

$$[\![\mathsf{N}et_k]\!] = [\![P \mid \ldots \mid P]\!] = [\![P]\!] \mid \ldots \mid [\![P]\!] = [\![\mathsf{N}et_1]\!] \mid \ldots \mid [\![\mathsf{N}et_1]\!]$$

By lemma 1 then there exists a maximal computation $\mathcal{C}_k$ of $[\![\mathsf{N}et_k]\!]$ such that $\mathrm{Obs}(\mathcal{C}_1) = \{y\} \subseteq \mathrm{Obs}(\mathcal{C}_k)$, while no computation of $\mathsf{N}et_k$ contains observable y for $k \geq 2$, which contradicts the weakly-observation-respecting property of the encoding function $[\![\cdot]\!]$.

Theorem 5. *There is no distribution-preserving and weakly-observation-respecting encoding of CPG into the π-Calculus.*

Proof. By lemma 4 exactly like for theorem 4.

Theorem 6. *There is no distribution-preserving and weakly-observation-respecting encoding of FAP into the bπ-Calculus.*

Proof. By lemmas 2 and 3, exactly like for theorem 4.

Theorem 7. *There is no distribution-preserving and weakly-observation-respecting encoding of CPG into the bπ-Calculus.*

Proof. By lemmas 2 and 4, exactly like for theorem 4.

5 Conclusion

We have considered FAP, a finite fragment of CCS added with global priority, and we have proved, by means of leader-election-based separation results, that it is not possible to encode it into CPG under uniformity and independence-preserving conditions on the encoding, thus providing the first expressiveness separation result between global and local priority within a process algebra framework. We have then proved that FAP cannot be distributively translated into the π-Calculus even if allowing partially correct implementations, i.e. encodings which may introduce divergence in computations or also premature termination caused by deadlock.

We have then analysed another setting, called last man standing (LMS) problem which allows to considerably strengthen the separation between prioritised (with both global or local priority) languages and non prioritised ones, by showing that even if we equip the language with broadcast-based primitives like the bπ-Calculus, the expressiveness of priority cannot be obtained under parallel-preserving conditions.

In conclusion we have showed that, within the context of the process algebras considered here, it is not possible to have a distribution-preserving encoding of neither global nor local priority in non-prioritised languages even if we admit asymmetric translations or divergence/failure in the computation as a consequence of livelocks or deadlocks. This impossibility result does not depend on the capability of communication of names or values, synchrony or asynchrony of the output, scope extrusion, choice on the available input/outputs, recursion or replication, point-to-point or broadcast communication/synchronisation type, but depends only on the power of instantaneous preemption characteristic of the prioritised languages considered in this paper. As a consequence we can see that it is not possible any purely distributed implementation of such kinds of priority on top of standard process calculi even if admitting good or randomised encodings like those considered in [20,21] for the implementation of the choice operator. The strength of the separation suggests also that any encoding trying to preserve some relaxed condition on the distribution may be affected by severe performance issues due to the further synchronisations needed to preserve the constraint of instantaneous preemption.

As future work we plan to analyse process algebras equipped with non-instantaneous priority (in the style of the expressiveness study on PrioLinCa [23]) i.e. languages where the effect of preemption is not immediate, in order to better characterise the expressive power of preemption and to identify prioritised constructs easier to implement in a parallel, if not distributed, framework.

References

1. Baeten, J.C.M., Bergstra, J.A., Klop, J.W.: Ready-trace semantics for concrete process algebra with the priority operator. Comput. J. 30(6), 498–506 (1987)
2. Camilleri, J., Winskel, G.: Ccs with priority choice. Inf. Comput. 116(1), 26–37 (1995)

3. Cleaveland, R., Hennessy, M.: Priorities in process algebras. Inf. Comput. 87(1/2), 58–77 (1990)
4. Cleaveland, R., Lüttgen, G., Natarajan, V.: Priority in process algebra. In: Bergstra, J., Ponse, A., Smolka, S. (eds.) Handbook of Process Algebra, pp. 711–765. Elsevier, Amsterdam (2001)
5. Phillips, I.: Ccs with priority guards. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 305–320. Springer, Heidelberg (2001)
6. Bernardo, M., Gorrieri, R.: Extended markovian process algebra. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 315–330. Springer, Heidelberg (1996)
7. Hermanns, H.: Interactive Markov Chains: The Quest for Quantified Quality. In: Hermanns, H. (ed.) Interactive Markov Chains. LNCS, vol. 2428, Springer, Heidelberg (2002)
8. Bravetti, M., Gorrieri, R.: The theory of interactive generalized semi-markov processes. Theor. Comput. Sci. 282(1), 5–32 (2002)
9. Bernardo, M., Gorrieri, R.: A tutorial on empa: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. Theor. Comput. Sci. 202(1-2), 1–54 (1998)
10. Baeten, J., Bergstra, J., Klop, J.: Syntax and defining equations for an interrupt mechanism in process algebra. Fundamenta Informaticae IX(2), 127–168 (1986)
11. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
12. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, i. Inf. Comput. 100(1), 1–40 (1992)
13. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, ii. Inf. Comput. 100(1), 41–77 (1992)
14. Ene, C., Muntean, T.: Expressiveness of point-to-point versus broadcast communications. In: Ciobanu, G., Păun, G. (eds.) FCT 1999. LNCS, vol. 1684, pp. 258–268. Springer, Heidelberg (1999)
15. Bougé, L.: On the existence of symmetric algorithms to find leaders in networks of communicating sequential processes. Acta Inf. 25(2), 179–201 (1988)
16. Palamidessi, C.: Comparing the expressive power of the synchronous and asynchronous pi-calculi. Mathematical Structures in Computer Science 13(5), 685–719 (2003)
17. Phillips, I.: Ccs with priority guards. Available at http://wwwhomes.doc.ic.ac.uk/~iccp/papers/ccspgfullrevised.pdf
18. Milner, R.: The polyadic pi-calculus: a tutorial. In: Bauer, F.L., Brauer, W., Schwichtenberg, H. (eds.) Logic and Algebra of Specification, pp. 203–246. Springer, Heidelberg (1993)
19. Milner, R.: Communicating and mobile systems: the π-calculus. Cambridge University Press, New York, NY, USA (1999)
20. Nestmann, U.: What is a "good" encoding of guarded choice? Inf. Comput. 156(1-2), 287–319 (2000)
21. Palamidessi, C., Herescu, O.M.: A randomized encoding of the pi-calculus with mixed choice. Theor. Comput. Sci. 335(2-3), 373–404 (2005)
22. Cardelli, L., Gordon, A.D.: Mobile ambients. In: Nivat, M. (ed.) ETAPS 1998 and FOSSACS 1998. LNCS, vol. 1378, pp. 140–155. Springer, Heidelberg (1998)
23. Bravetti, M., Gorrieri, R., Lucchi, R., Zavattaro, G.: Quantitative information in the tuple space coordination model. Theor. Comput. Sci. 346(1), 28–57 (2005)

A Marriage of Rely/Guarantee and Separation Logic

Viktor Vafeiadis and Matthew Parkinson

University of Cambridge

Abstract. In the quest for tractable methods for reasoning about concurrent algorithms both rely/guarantee logic and separation logic have made great advances. They both seek to tame, or control, the complexity of concurrent interactions, but neither is the ultimate approach. Rely-guarantee copes naturally with interference, but its specifications are complex because they describe the entire state. Conversely separation logic has difficulty dealing with interference, but its specifications are simpler because they describe only the relevant state that the program accesses.

We propose a combined system which marries the two approaches. We can describe interference naturally (using a relation as in rely/guarantee), and where there is no interference, we can reason locally (as in separation logic). We demonstrate the advantages of the combined approach by verifying a lock-coupling list algorithm, which actually disposes/frees removed nodes.

1 Introduction

Reasoning about shared variable concurrent programs is difficult, because the interference between the simultaneously executing threads must be taken into account. Our aim is to find methods that allow this reasoning to be done in a modular and composable way.

On the one hand, we have rely/guarantee, a well-established method, introduced by Jones [11], that is popular in the derivation and the post-hoc verification of concurrent algorithms. RG provides a good way of describing interference by having two relations, the rely R and the guarantee G, which describe the state changes performed by the environment or by the program respectively. Its disadvantage is that the specification of interference is *global*: it must be checked against every state update, even if it is 'obvious' that the update cannot interfere with anything else. Even Jones [12] acknowledges this limitation and still considers the search for a satisfactory compositional approach to concurrency an 'open problem.'

On the other hand, the recent development of separation logic [19,15] suggests that greater modularity is possible. There, the $*$ operator and the frame rule are used to carve all irrelevant state out of the specification and focus only on the state that matters for the execution of a certain component or thread. This makes specifications *local*; two components may interfere, only if they have overlapping specifications. Its disadvantage is that, in dealing with concurrent programs, it took the simplest approach and uses invariants to specify thread interaction. This makes expressing the relational nature of interference often quite difficult and requires many auxiliary variables [17]. Even O'Hearn acknowledges the weaknesses of separation logic, and asks if "a marriage between separation logic and rely-guarantee is also possible" [15].

Here we present such a marriage of rely/guarantee and separation logic, which combines their advantages and eliminates some of their weaknesses. We split the state into two disjoint parts: (i) the shared state which is accessible by all threads, and (ii) the local state which is accessible by a single component. Then, we use rely/guarantee to deal with the shared state, and separation logic to deal with the local state. This is best illustrated by our parallel composition rule:

$$\frac{\vdash C_1 \ \mathbf{sat}\ (p_1, R \cup G_2, G_1, q_1) \qquad \vdash C_2 \ \mathbf{sat}\ (p_2, R \cup G_1, G_2, q_2)}{\vdash C_1 \| C_2 \ \mathbf{sat}\ (p_1 * p_2, R, G_1 \cup G_2, q_1 * q_2)}$$

This rule is identical to the standard rely/guarantee rule except for the use of $*$ instead of $\wedge$ in the pre- and post-conditions. In our specifications, the preconditions (e.g. p_1) and the postconditions (e.g. q_1) describe both the local and the shared state. The rely conditions (e.g. $R \cup G_2$) and the guarantee conditions (e.g. G_1) describe inter-thread interference: how the shared state gets modified.

The separating conjunction between assertions about both the local and the shared state splits local state (l) in two parts, but does not divide the shared state (s).

$$(p_1 * p_2)(l, s) \overset{\mathbf{def}}{=} \exists l_1\, l_2.\ l = l_1 \uplus l_2 \wedge p_1(l_1, s) \wedge p_2(l_2, s)$$

The parallel composition rules of rely/guarantee and separation logic are special cases of our parallel composition rule. (1) When the local state is empty, then $p_1 * p_2 = p_1 \wedge p_2$ and we get the standard rely/guarantee rule. (2) When the shared state is empty, we do not need to describe its evolution (R and G are the identity relation). Then $p_1 * p_2$ has the same meaning as separation logic $*$, and we get the parallel rule of concurrent separation logic without resource invariants (see §2.1).

An important aspect of our approach is that the boundaries between the local state and the shared state are not fixed, but may change as the program runs. This "ownership transfer" concept is fundamental to proofs in concurrent separation logic.

In addition, as we encompass separation logic, we can cleanly reason about dynamically allocated data structures and explicit memory management, avoiding the need to rely on a garbage-collector. In §4, we demonstrate this by verifying a lock-coupling list algorithm, which actually disposes/frees removed nodes.

2 Technical Background

In this paper, we reason about a parallel programming language with pointer operations. Let x, y and z range over logical variables, and $\mathtt{x}$, $\mathtt{y}$ and $\mathtt{z}$ over program variables. We assume tid is a special variable that identifies the current thread. Commands C and expressions e are given by the following grammar,

$$\begin{aligned} C ::= {} & \texttt{x:=}e \mid \texttt{x:=}[e] \mid [e_1]\texttt{:=}e_2 \mid \texttt{x:=}\mathrm{cons}(e_1, \ldots, e_n) \mid \mathrm{dispose}(e) \\ & \mid C_1; C_2 \mid C_1 \| C_2 \mid \mathbf{if}(b)\{C_1\}\,\mathbf{else}\,\{C_2\} \mid \mathbf{while}(b)\{C\} \mid \mathbf{atomic}(b)\{C\} \\ e ::= {} & x \mid \mathrm{x} \mid e + e \mid n \end{aligned}$$

where b ranges over boolean expressions. Note that expressions e are *pure*: they do not refer to the heap. In the grammar, each assignment contains at most one heap access; assignments with multiple heap accesses can be performed using multiple assignments and temporary variables to store the intermediate results.

The semantics of **atomic** are that C will be executed in one indivisible step. This could be implemented through locking, hardware atomicity, transactional memories, etc. Choosing **atomic** over a given synchronisation primitive (e.g. locks) enables our reasoning to be applied at multiple abstraction levels. In any case, any synchronisation primitive can be encoded using **atomic**.

2.1 Local Reasoning – Separation Logic

In Hoare logic [9], assertions describe properties of the *whole* memory, and hence specifications, e.g. $\{P\}\ C\ \{Q\}$, describe a change of the whole memory. This is inherently *global reasoning*. Anything that is not explicitly preserved in the specification could be changed, for example $\{\texttt{x} = 4\}$ `y:=5` $\{\texttt{x} = 4\}$. Here `y` is allowed to change, even though it is not mentioned in the specification.[1]

The situation is different in *separation logic* [19]. Assertions describe properties of *part* of the memory, and hence specifications describe changes to *part* of the memory. The rest of the memory is guaranteed to be unchanged. This is the essence of *local reasoning*, specifications describe only the memory used by a command, its footprint.

The strength of separation logic comes from a new logical connective: the separating conjunction, $*$. $P * Q$ asserts the state can be split into two parts, one described by P and the other by Q. The separating conjunction allows us to formally capture the essence of *local reasoning* with the following rules:

$$\frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}\ \text{(Frame)} \qquad \frac{\{P_1\}\ C_1\ \{Q_1\} \quad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \| C_2\ \{Q_1 * Q_2\}}\ \text{(Par)}$$

The first rule says, if P is separate from R, and C transforms P into Q then if C finishes we have Q and separately still have R. The second rule says that if two threads have disjoint memory requirements, they can execute safely in parallel, and the postcondition is simply the composition of the two threads' postconditions.[2]

[1] 'Modifies clauses' solve this problem, but they are neither pretty nor general.

[2] Originally, separation logic did not consider global variables as resource; hence the proof rules had nasty side-conditions. Later, this problem was solved by Bornat et al. [2]. By disallowing direct assignments to global variables, we avoid the problem.

Separation logic has the following assertions for describing the heap, h:

$$P, Q, S ::= \textbf{false} \mid \textbf{emp} \mid e = e' \mid e \mapsto e' \mid \exists x.\, P \mid P \Rightarrow Q \mid P * Q \mid P \mathrel{-\!\!\circledast} Q$$

We encode $\neg, \wedge, \vee, \forall$, and **true** in the classical way. **emp** stands for the empty heap; $e \mapsto e'$ for the heap consisting of a single cell with address e and contents e'. Separating conjunction, $P{*}Q$, is the most important operator of separation logic. A heap h satisfies $P{*}Q$, if it can be split in two parts, one of which satisfies P and the other satisfies Q. There remains one new connective to describe: *septraction*, $P \mathrel{-\!\!\circledast} Q$.[3] Intuitively, $P \mathrel{-\!\!\circledast} Q$ represents removing P from Q. Formally, it means the heap can be extended with a state satisfying P, and the extended state satisfies Q.

$$\begin{aligned} h, i \vDash_{\text{SL}} (P * Q) &\overset{\textbf{def}}{=} \exists h_1, h_2.\, (h_1 \uplus h_2 = h) \wedge h_1, i \vDash_{\text{SL}} P \wedge h_2, i \vDash_{\text{SL}} Q \\ h, i \vDash_{\text{SL}} (P \mathrel{-\!\!\circledast} Q) &\overset{\textbf{def}}{=} \exists h_1, h_2.\, (h_1 \uplus h = h_2) \wedge h_1, i \vDash_{\text{SL}} P \wedge h_2, i \vDash_{\text{SL}} Q \end{aligned}$$

Finally, $e \mapsto e_1, \ldots, e_n$ is a shorthand for $(e \mapsto e_1) * \ldots * (e + n - 1 \mapsto e_n)$.

3 The Combined Logic

3.1 Describing Interference

The strength of rely/guarantee is the careful description of interference between parallel processes. We describe interference in terms of actions $P \leadsto Q$ which describe the changes performed to the shared state. These resemble Morgan's *specification statements* [13], and P and Q will typically be linked with some existentially quantified logical variables. (We do not need to mention separately the set of modified shared locations, because these are all included in P.)

The meaning of an action $P \leadsto Q$ is that it replaces the part of the state that satisfies P before the action with a part satisfying Q. Its semantics is the following relation:

$$[\![P \leadsto Q]\!] = \{(h_1 \uplus h_0, h_2 \uplus h_0) \mid h_1, i \vDash_{\text{SL}} P \wedge h_2, i \vDash_{\text{SL}} Q\}$$

It relates some initial state h_1 satisfying the precondition P to a final state h_2 satisfying the postcondition. In addition, there may be some disjoint state h_0 which is not affected by the action. In the spirit of separation logic, we want action specifications as 'small' as possible, describing h_1 and h_2 but not h_0, and use the frame rule to perform the same update on a larger state.

The rely and guarantee conditions are simply sets of actions. Their semantics as a relation is the reflexive and transitive closure of the union of the semantics of each action in the set. We shall write R for a syntactic rely condition (i.e. a set of actions) and $\mathcal{R}$ for a semantic rely condition (i.e. a binary relation).

[3] Sometimes called "existential magic wand", as it is the dual to "magic wand": $P \mathrel{-\!\!\circledast} Q \overset{\textbf{def}}{=} \neg(P \mathrel{-\!\!*} \neg Q)$. It has been used in the connection with modal logic in [4].

3.2 Stability

Rely/guarantee reasoning requires that every pre- and post-condition in a proof is stable under environment interference. An assertion S is stable under interference of a relation $\mathcal{R}$ if and only if whenever S holds initially and we perform an update satisfying $\mathcal{R}$ then the resulting state still satisfies S.

Definition 1 (Stability). *$S;\mathcal{R} \Longrightarrow S$ iff for all s, s' and i such that $s, i \vDash_{\mathrm{SL}} S$ and $(s, s') \in \mathcal{R}$, then $s', i \vDash_{\mathrm{SL}} S$*

By representing the interference $\mathcal{R}$ as a set of actions, we reduce stability to a simple syntactic check. For a single action $[\![P \rightsquigarrow Q]\!]$, the following separation logic implication is necessary and sufficient:

Lemma 1. *$S; [\![P \rightsquigarrow Q]\!] \Longrightarrow S \quad$ iff $\quad \vDash_{\mathrm{SL}} (P -\!\circledast S) * Q \implies S$.*

Informally, it says that if from a state that satisfies S, we subtract the part of the state satisfying P, and replace it with some state satisfying Q, then the result should still satisfy S. When the action cannot fire because there is no substate of S satisfying P, then $P -\!\circledast S$ is false and the implication holds trivially.

An assertion S is stable under interference of a set of actions R when it is stable under interference of every action in R.

Lemma 2. *$S; (\mathcal{R}_1 \cup \mathcal{R}_2)^* \Longrightarrow S \quad$ iff $\quad S; \mathcal{R}_1 \Longrightarrow S$ and $S; \mathcal{R}_2 \Longrightarrow S$.*

Finally, we define $\mathrm{wssa}_{\mathcal{R}}(Q)$ to be the weakest assertion that is stronger than Q and stable under $\mathcal{R}$.

Definition 2 (Weakest stable stronger assertion). *(1) $wssa_{\mathcal{R}}(Q) \Rightarrow Q$,*
(2) $wssa_{\mathcal{R}}(Q); \mathcal{R} \Longrightarrow wssa_{\mathcal{R}}(Q)$, and
(3) for all P, if $P; \mathcal{R} \Longrightarrow P$ and $P \Rightarrow Q$, then $P \Rightarrow wssa_{\mathcal{R}}(Q)$.

3.3 Local and Shared State Assertions

We can specify a state using two assertions, one describing the local state and the other the shared state. However, this approach has some drawbacks: specifications are longer, and extending the logic to a setting with multiple disjoint regions of shared state is clumsy.

Instead, we consider a unified assertion language that describes both the local and the shared state. This is done by extending the positive fragment of separation logic assertions with 'boxed' terms. We could use boxes for both local and shared assertions: for example, $\boxed{P}_{\mathrm{local}}$ and $\boxed{P}_{\mathrm{shared}}$. However, since $\boxed{P}_{\mathrm{local}} * \boxed{Q}_{\mathrm{local}} \iff \boxed{P * Q}_{\mathrm{local}}$ holds for $*$, and all the classical operators, we can omit the $\boxed{\_}_{\mathrm{local}}$ and the "$_{\mathrm{shared}}$" subscript. Hence the syntax of assertions is

$$p, q, r ::= P \mid \boxed{P} \mid p * q \mid p \wedge q \mid p \vee q \mid \exists x.\, p \mid \forall x.\, p$$

Semantically, we split the state, σ, of the system into two components: the local state l, and the shared state s. Each component state may be thought to be a partial finite function from locations to values. We require that the domains of the two states are disjoint, so that the total state is simply the (disjoint) union of the two states. Assertions without boxes describe purely the local state l, whereas a boxed assertion $\boxed{P}$ describes the shared state s. Formally, we give the semantics with respect to a 'rely' condition R, a set of actions describing the environment interference:

$$\begin{array}{lcl}
l, s, i \vDash_R P & \iff & l, i \vDash_{\mathrm{SL}} P \\
l, s, i \vDash_R \boxed{P} & \iff & l = \emptyset \wedge s, i \vDash_{\mathrm{SL}} \mathrm{wssa}_{[\![R]\!]}(P) \\
l, s, i \vDash_R p_1 * p_2 & \iff & \exists l_1, l_2.\ (l = l_1 \uplus l_2) \wedge (l_1, s, i \vDash_R p_1) \wedge (l_2, s, i \vDash_R p_2) \\
l, s, i \vDash_R p_1 \wedge p_2 & \iff & (l, s, i \vDash_R p_1) \wedge (l, s, i \vDash_R p_2) \\
\ldots & &
\end{array}$$

Note that $*$ is multiplicative over the local state, but additive over the shared state. Hence, $\boxed{P} * \boxed{Q} \implies \boxed{P \wedge Q}$. The semantics of shared assertions, $\boxed{P}$, could alternatively be presented without $l = \emptyset$. This results in an equally expressive logic, but the definition above leads to shorter assertions in practice.

We use $\mathrm{wssa}_{[\![R]\!]}(\_)$ to make assertions semantically resistant to interference:

Lemma 3. *If* $(l, s, i \vDash_R p)$, $s' \uplus l$ *defined and* $[\![R]\!](s, s')$ *then* $(l, s', i \vDash_R p)$.

We define an assertion to be syntactically stable if each of the assertions about the shared state is stable. By construction, any assertion about the local state of a component is unaffected by other components, because interference can happen only on the shared state. On the other hand, a boxed assertion $\boxed{S}$ may be affected.

Definition 3 (Stable assertion). *P stable under R always; $\boxed{P}$ stable under R iff $P; [\![R]\!] \implies P$; $(p\ op\ q)$ stable under R iff p stable under R and q stable under R; and $(qu\, x.\ p)$ stable under R iff p stable under R where $op ::= \wedge \mid \vee \mid *$ and $qu ::= \forall \mid \exists$.*

This syntactic condition allows us to change the interpretation of a formula to a more permissive rely.

Lemma 4. *If* $(l, s, i \vDash_R p)$, $[\![R]\!] \subseteq [\![R']\!]$ *and p stable under R' then* $(l, s, i \vDash_{R'} p)$.

We present a few entailments for formulae involving shared states.

$$\frac{P \vdash_{\mathrm{SL}} Q}{\boxed{P} \vdash \boxed{Q}} \qquad \boxed{P} \wedge \boxed{Q} \vdash \boxed{P \wedge Q} \qquad \boxed{P} \vee \boxed{Q} \vdash \boxed{P \vee Q} \qquad \boxed{P} * \boxed{Q} \vdash \boxed{P \wedge Q}$$

$$\forall x.\ \boxed{P} \vdash \boxed{\forall x.\ P} \qquad \exists x.\ \boxed{P} \vdash \boxed{\exists x.\ P} \qquad \boxed{P} \vdash \boxed{P} * \boxed{P} \qquad \boxed{P} \vdash \mathbf{emp}$$

3.4 Ownership Transfer

Usually the precondition and postcondition of an action have the same heap footprint. For example, consider the action saying that x can be incremented:

$$\mathtt{x} \mapsto M \quad \rightsquigarrow \quad \mathtt{x} \mapsto N \wedge N \geq M \qquad \text{(Increment)}$$

If they have a different footprints, this indicates a transfer of ownership between the shared state and the local state of a thread. Consider a simple lock with

$$\frac{\vdash C \textbf{ sat } (p, R, G, q) \quad \left(\begin{array}{l}(r \text{ stable under } R \cup G)\\ \vee\ (C \text{ has no } \textbf{atomics})\end{array}\right)}{\vdash C \textbf{ sat } (p * r, R, G, q * r)} \qquad \frac{Q = (P * X \mapsto Y) \quad x \notin fv(P)}{\vdash (x := [e]) \textbf{ sat } (\boxed{Q} \wedge e = X, R, G, \boxed{Q} \wedge x = Y)}$$

$$\frac{\vdash C_1 \textbf{ sat } (p, R, G, q) \quad \vdash C_2 \textbf{ sat } (q, R, G, r)}{\vdash C_1; C_2 \textbf{ sat } (p, R, G, r)} \qquad \frac{\begin{array}{ll}\vdash C_1 \textbf{ sat } (p_1, R \cup G_2, G_1, q_1) & p_1 \text{ stable under } R \cup G_1\\ \vdash C_2 \textbf{ sat } (p_2, R \cup G_1, G_2, q_2) & p_2 \text{ stable under } R \cup G_2\end{array}}{\vdash C_1 \| C_2 \textbf{ sat } (p_1 * p_2, R, G_1 \cup G_2, q_1 * q_2)}$$

$$\frac{\begin{array}{ll}\vdash C \textbf{ sat } (P_1 * P_2, \{\}, \{\}, Q_1 * Q_2) & \boxed{Q} \text{ stable under } R\\ \overline{y} \cap FV(P_2) = \emptyset \quad P \Rightarrow P_1 * F \quad Q_1 * F \Rightarrow Q & (P_1 \rightsquigarrow Q_1) \subseteq G\end{array}}{\vdash (\textbf{atomic}\{C\}) \textbf{ sat } (\boxed{\exists \overline{y}.\ P} * P_2, R, G, \exists \overline{y}.\ \boxed{Q} * Q_2)}$$

Fig. 1. Proof rules

two operations: (Acq) which changes the lock bit from 0 to 1, and removes the protected object, $list(y)$, from the shared state; and (Rel) which changes the lock bit from 1 to 0, and replaces the protected object into the shared state. We can represent these two operations formally as

$$(\mathsf{x} \mapsto 0) * list(\mathsf{y}) \rightsquigarrow \mathsf{x} \mapsto 1 \quad (\mathsf{Acq}) \qquad\qquad \mathsf{x} \mapsto 1 \rightsquigarrow (\mathsf{x} \mapsto 0) * list(\mathsf{y}) \quad (\mathsf{Rel})$$

3.5 Specifications and Proof Rules

The judgement $\vdash C$ **sat** (p, R, G, q) semantically says that any execution of C from an initial state satisfying p and under interference at most R, (i) does not fault (e.g. access unallocated memory), (ii) causes interference at most G, and, (iii) if it terminates, its final state satisfies q.

The key proof rules are presented in Figure 1. The rest can be found in the technical report [22]. From separation logic, we inherit the frame rule. This rule says that a program safely running with initial state p can also be executed with additional state r. As the program runs safely without r, it cannot access r when it is present; hence, r is still true at the end. The additional premise is needed because r might mention the shared state and C might modify it in an **atomic**.

We adopt all of the small axioms for local state from separation logic (not presented) [14]. Additionally, we have a read axiom (Fig. 1 top right) for shared state, which allows a non-atomic read from a shared location if we can rely on its value not changing. Note that we do not need to check stability for this read.

The next rule is that of conditional critical regions **atomic**$(b)\{C\}$. For clarity, we present the rule where the guard b is just **true**. The general case, where b is non-trivial and may access the heap, just complicates the essential part of the rule. A simple rule for critical regions would be the following:

$$\frac{\vdash C \textbf{ sat } (P, \{\}, \{\}, Q) \quad (P \rightsquigarrow Q) \subseteq G \quad \boxed{Q} \text{ stable under } R}{\vdash (\textbf{atomic}\{C\}) \textbf{ sat } (\boxed{P}, R, G, \boxed{Q})}$$

$$\frac{}{x \mapsto y \rightsquigarrow x \mapsto y \subseteq G}\ \textsc{G-Exact} \qquad \frac{P \rightsquigarrow Q \in G}{P \rightsquigarrow Q \subseteq G}\ \textsc{G-Ax}$$

$$\frac{P_1 \rightsquigarrow S * Q_1 \subseteq G \quad P_2 * S \rightsquigarrow Q_2 \subseteq G}{P_1 * P_2 \rightsquigarrow Q_1 * Q_2 \subseteq G}\ \textsc{G-Seq} \qquad \frac{P \rightsquigarrow Q \subseteq G}{P[e/x] \rightsquigarrow Q[e/x] \subseteq G}\ \textsc{G-Sub}$$

$$\frac{\vDash_{\mathrm{SL}} P' \Rightarrow P \quad P \rightsquigarrow Q \subseteq G \quad \vDash_{\mathrm{SL}} Q' \Rightarrow Q}{P' \rightsquigarrow Q' \subseteq G}\ \textsc{G-Cons} \qquad \frac{(P * F) \rightsquigarrow (Q * F) \subseteq G}{P \rightsquigarrow Q \subseteq G}\ \textsc{G-CoFrm}$$

Fig. 2. Rules and axioms for guarantee allows an action

As in RG, we must check that the postcondition is stable under interference from the environment, and that changing the shared state from P to Q is allowed by the guarantee G.

This rule is sound, but too weak in two ways. First, it does not allow critical regions to access any local state, as the precondition $\boxed{P}$ requires that the local state is empty. Second, it requires that the critical region changes the *entire* shared state from P to Q and that the guarantee condition allows such a change. Thus, we extend the rule by (i) adding a precondition P_2 and a postcondition Q_2 for the local state, and (ii) allowing the region to change a part P_1 of P into a part Q_1 of Q, ensuring that the rest F does not change. Additionally, we allow some existential quantifiers, $\overline{y}$ in the shared state to be pulled out over both the shared and local state.

A specification, $P_1 \rightsquigarrow Q_1$ is allowed by a guarantee G if its effect is contained in G. Fig. 2 provides rules to approximate this definition in proofs. The rule G-Seq allows actions to be sequenced and builds in a form of framing. Note that, if S is empty, then the rule is a parallel composition of two actions; if P_2 and Q_1 are empty, then the rule sequences the actions. It would be simpler, if we simply included the frame rule however this is unsound. In fact, the coframe rule G-CoFrm is admissible. G-Cons is similar to the rule of consequence, but the second implication is reversed, $Q \Rightarrow Q'$. Semantically, the property is defined as follows:

Definition 4. $P \rightsquigarrow Q \subseteq G$ *iff* $[\![P \rightsquigarrow Q]\!] \subseteq [\![G]\!]$.

There is a side-condition to the atomic rule requiring that Q is a *precise* assertion. This is formally defined in §5 (Footnote. 7). This is a technical requirement inherited from concurrent separation logic. It ensures that the splitting of the resultant state into local and shared portions is unambiguous.

We reiterate the parallel composition rule from the introduction. As the interference experienced by thread C_1 can arise from C_2 or the environment of the parallel composition, we have to ensure that this interference $R \cup G_2$ is allowed. Similarly C_2 must be able to tolerate interference from C_1 and from the environment of the parallel composition. The precondition and postcondition of the composition are the separating conjunction, $*$, of the preconditions/postconditions of the individual threads. In essence, this is the conjunction of the shared

```
lock(p) {
 atomic(p.lock==0){
   p.lock = tid;
//p.oldn = p.next;
 }
}
unlock(p) {
 atomic(true) {
   p.lock = 0;
 }
}
```

```
locate(e) {
 local p,c;
 p = Head;
 lock(p);
 c = p.next;
 while(c.value<e){
  lock(c);
  unlock(p);
  p = c;
  c = p.next;
 }
 return (p,c);
}
```

```
add(e) {
 local x,y,z;
 (x,z)=locate(e);
 if(z.value!=e){
  y = cons(0,e,z);
  x.next = y;
 }
 unlock(x);
}
```

```
remove(e) {
 local x,y,z;
 (x,y)=locate(e);
 if(y.value==e){
  lock(y);
  z = y.next;
  x.next = z;
  unlock(x); // A
  dispose(y);
 } else {
  unlock(x);
 }
}
```

Fig. 3. Source code for lock coupling list operations. For clarity, we use a field notation, hence we encode p.lock, x.value, x.next and p.oldn as [p], [x + 1], [x + 2] and [p + 3], respectively. Commented code is auxiliary, that is, required only for the proof.

state assertions, and the separating conjunction of the local state assertions (cf. the semantics of $*$ in §3.3).

The proof rules for conditional and iterative commands are completely standard (See [22].)

4 Example

This section uses the new logic to verify a fine-grained concurrent linked list implementation of a mutable set data structure (see Fig. 3). It has operations add which adds an element to the set, and remove which removes an element from the set.

The algorithm associates one lock per list node rather than have a single lock for the entire list. Traversing the list uses *lock coupling*: the lock on one node is not released until the next node is locked. Somewhat like a person climbing a rope "hand-over-hand," you always have at least one hand on the rope.

An element is added to the set by inserting it in the appropriate position, while holding the lock of its previous node. It is removed by redirecting the previous node's pointer, while both the previous and the current node are locked. This ensures that deletions and insertions can happen concurrently in the same list. The algorithm makes two assumptions about the list: (1) it is sorted; and (2) the first and last elements have values $-\infty$ and $+\infty$ respectively. This allows us to avoid checking for the end of the list.

Node predicates. We use three predicates to represent a node in the list: (1) $N_s(x, v, y)$, for a node at location x with contents v and tail pointer y and with the lock status set to s; (2) $U(x, v, y)$ for an unlocked node at location x withcontents v and tail pointer y; and (3) $L_t(x, v, y)$ for a node locked with thread identifier t. We use $N_{\_}(x, v, y)$ for a node that may or may not be locked.

$$N_s(x,v,y) \overset{\textbf{def}}{=} x \mapsto s, v * \begin{pmatrix} (s = 0 \wedge x+2 \mapsto y, \_) \\ \vee\ (s \neq 0 \wedge x+3 \mapsto y) \end{pmatrix} \wedge x \bmod 4 = 0$$
$$U(x,v,y) \overset{\textbf{def}}{=} N_0(x,v,y) \qquad\qquad L_t(x,v,y) \overset{\textbf{def}}{=} N_t(x,v,y) \wedge t > 0$$

We assume nodes are aligned, $x \bmod 4 = 0$, and **cons** returns aligned nodes.[4] The thread identifier parameter in the locked node is required to specify that a node can only be unlocked by the thread that locked it. The fourth field/cell is auxiliary. It is used to store the last value of the nodes tail before it was locked. Once a node is locked its tail field is released to the locking thread, allowing it to mutate the field outside of critical sections, the auxiliary field is used in the proof to track the list structure.

Actions. The algorithm does four kinds of actions: (1) lock, which locks a node, (2) unlock, which unlocks a node, (3) add, which inserts a new node to the list, and (4) delete, which removes a node from the list. All of these actions are parameterised with a set of thread identifiers, T. This allows us to use the actions to represent both relies and guarantees. In particular, we take a thread with identifier tid to have the guarantee with $T = \{\text{tid}\}$, and the rely to use the complement of this set. Let $I(T)$ be the set of these four actions.

The first two actions are straightforward:

$$t \in T \wedge U(x,v,n) \rightsquigarrow L_t(x,v,n) \qquad \text{(lock)}$$
$$t \in T \wedge L_t(x,v,n) \rightsquigarrow U(x,v,n) \qquad \text{(unlock)}$$

Now, consider adding a node to the list. We begin by describing an action that ignores the sorted nature of the list:

$$t \in T \wedge L_t(x,u,n) \rightsquigarrow L_t(x,u,m) * U(m,v,n)$$

To add an element to the list, we must have locked the previous node, and then we can swing the tail pointer to the added node. The added node must have the same tail as previous node before the update. To preserve the sorted order of the list, the actual add action must also mention the next node: the inserted value must be between the previous and the next values.

$$(t \in T) \wedge (u < v < w) \wedge (L_t(x,u,n) * N_s(n,w,y)) \\ \rightsquigarrow L_t(x,u,m) * U(m,v,n) * N_s(n,w,y) \qquad \text{(add)}$$

The final action we allow is removing an element from the list. We must lock the node we wish to delete, n, and its previous node, x. The tail of the previous node must be updated to the deleted node's tail, m.

$$(v < \infty) \wedge (t \in T) \wedge (L_t(x,u,n) * L_t(n,v,m)) \rightsquigarrow L_t(x,u,m) \qquad \text{(delete)}$$

[4] Without this restriction a node could be formed by parts of two adjacent nodes. Instead of assuming alignment, this problem can also be solved by allowing contexts in actions, for example the node is reachable from the head.

List predicate. We use separation to describe the structure of the shared list. The predicate $ls(x, A, y)$ describes a list segment starting at location x with the final tail value of y, and with contents A. We use $\cdot$ as a list separator.

$$ls(x, \emptyset, x) \overset{\mathbf{def}}{=} \mathbf{emp} \qquad ls(x, v{\cdot}B, y) \overset{\mathbf{def}}{=} (\exists z.\ x \neq y \land N\_(x, v, z) * ls(z, B, y))$$

Note, as we use separation logic we do not need any reachability predicates, our predicate is simply a recursively defined predicate. The use of $*$ and the inequality $x \neq y$ ensures the list is acyclic. Removing a node from a list segment simply gives two list segments.

Proposition 1. *$(N_s(x, v, y) -\!\circledast\, ls(w, A, z))$ is equivalent to $\exists BC.\,(A = B{\cdot}v{\cdot}C) \land w \neq z \land \big(ls(w, B, x)\lfloor_z * ls(y, C, z)\lfloor_x\big)$ where $P\lfloor_x \overset{\mathbf{def}}{=} P \land \neg(x \mapsto \_ * true)$*

The algorithm works on sorted lists with the first and last values being $-\infty$ and $+\infty$ respectively. $s(A)$ represents this restriction on a logical list A.

$$srt(+\infty{\cdot}\epsilon) \overset{\mathbf{def}}{=} \mathbf{emp} \qquad srt(a{\cdot}b{\cdot}A) \overset{\mathbf{def}}{=} srt(b{\cdot}A) \land a < b \qquad s(-\infty{\cdot}A) \overset{\mathbf{def}}{=} srt(A)$$

Main proof. Appendix A contains the proof outline for the remove function. The outline presents the intermediate assertions in the proof. We present one step of the verification of remove function in detail: the unlock action labelled "A" in Fig. 3. For simplicity, we inline the unlock body.

$$\{\boxed{\exists AB.\ ls(\mathrm{Head}, A, \mathrm{x}) * L_{\mathrm{tid}}(\mathrm{x}, u, \mathrm{y}) * L_{\mathrm{tid}}(\mathrm{y}, \mathrm{e}, \mathrm{z}) * ls(\mathrm{z}, B, \mathrm{nil}) * s(A{\cdot}u{\cdot}B)} * (\mathrm{x}{+}2 \mapsto \mathrm{z})\}$$
$$\mathbf{atomic}\{\{L_{\mathrm{tid}}(\mathrm{x}, u, \mathrm{y}) * L_{\mathrm{tid}}(\mathrm{y}, \mathrm{e}, \mathrm{z}) * (\mathrm{x}{+}2 \mapsto \mathrm{z})\}\mathrm{x.lock} = 0; \{U(\mathrm{x}, u, \mathrm{z}) * L_{\mathrm{tid}}(\mathrm{y}, \mathrm{e}, \mathrm{z})\}\}$$
$$\{\boxed{\exists A.\ ls(\mathrm{Head}, A, \mathrm{nil}) * s(A)} * L_{\mathrm{tid}}(\mathrm{y}, \mathrm{e}, \mathrm{z})\}$$

We must prove four things: (1) the body meets its specification; (2) the body's specification is allowed by the guarantee; (3) the outer specification's postcondition is stable; and (4) find a frame, F, that satisfies the two implications.

The first is a simple proof in separation logic. The second follows as:

$$\frac{\begin{array}{c} L_{\mathrm{tid}}(\mathrm{x}, u, \mathrm{y}) * L_{\mathrm{tid}}(\mathrm{y}, \mathrm{e}, \mathrm{z}) \leadsto L_{\mathrm{tid}}(\mathrm{x}, u, \mathrm{z}) \subseteq I(\{\mathrm{tid}\}) \\ L_{\mathrm{tid}}(\mathrm{x}, u, \mathrm{z}) \leadsto U(\mathrm{x}, u, V z) \subseteq I(\{\mathrm{tid}\}) \end{array}}{L_{\mathrm{tid}}(\mathrm{x}, u, \mathrm{y}) * L_{\mathrm{tid}}(\mathrm{y}, \mathrm{e}, \mathrm{z}) \leadsto U(\mathrm{x}, u, \mathrm{z}) \subseteq I(\{\mathrm{tid}\})}\ \text{G-SEQ}$$

Third, to show $\boxed{\exists A.\ ls(\mathrm{Head}, A, \mathrm{nil}) * s(A)}$ is stable, we use Lemma 1 for the four actions in the rely: lock, unlock, add and delete. The proof of stability is long (hence omitted), but the proof steps are largely automatic. We can automate these checks [6].

Finally, we define F as $ls(\mathrm{Head}, A, \mathrm{x}) * ls(\mathrm{z}, B, \mathrm{nil}) * s(A{\cdot}u{\cdot}B)$

Theorem 1. *The algorithm in Fig. 3 is safe and keeps the list always sorted.*

5 Semantics and Soundness

Our semantics follows the abstract semantics for separation logic of Calcagno, O'Hearn and Yang [5]. Rather than presenting the semantics with respect to a

$$\frac{l \uplus s = l_1 \quad b(l_1) \quad l' \uplus s' = l_2 \quad Q(s') \quad (C, (l_1, \emptyset, o)) \xrightarrow{\mathbf{Emp}}{}^{*} (\mathbf{skip}, (l_2, \emptyset, o'))}{(\mathbf{atomic}_Q(b)\{C\}, (l, s, o)) \xrightarrow[\mathbf{p}]{\mathcal{R}} (\mathbf{skip}, (l', s', o'))} \qquad \frac{(C_1, \sigma) \xrightarrow[\mathbf{p}]{\mathcal{R}} (C_1', \sigma')}{(C_1 \| C_2, \sigma) \xrightarrow[\mathbf{p}]{\mathcal{R}} (C_1' \| C_2, \sigma')}$$

$$\frac{A(l, l') \quad (l', s, o) \in \mathsf{Heaps}}{(A, (l, s, o)) \xrightarrow[\mathbf{p}]{\mathcal{R}} (\mathbf{skip}, (l', s, o))} \qquad \frac{(\neg \exists l'.\ A(l, l'))}{(A, (l, s, o)) \xrightarrow[\mathbf{p}]{\mathcal{R}} \mathbf{fault}}$$

$$\frac{\mathcal{R}(s, s') \quad (l, s', o') \in \mathsf{Heaps}}{(C, (l, s, o)) \xrightarrow[\mathbf{e}]{\mathcal{R}} (C, (l, s', o'))}$$

Fig. 4. Abridged operational semantics

particular model of the heap, we use a partial commutative cancellative[5] monoid $(M, \uplus, \emptyset)$ as an abstract notion of a heap. We use m, l, s and o to range over elements of M.

Our logic explicitly deals with the separation between a thread's own local state (l) and the shared state (s), and hence implicitly the environment's own state (o). Our semantics are given with respect to a structured heap, which separates these three components.[6] This splitting is only used to prove the soundness of the logic. There is an obvious erasure to a semantics without a splitting.

Definition 5 (Structured heaps). *Heaps* $\overset{def}{=} \{(l, s, o) \mid \{l, s, o\} \subseteq M \wedge l \uplus s \uplus o$ *is defined*$\}$ *and* $(l_1, s_1, o_1) \uplus (l_2, s_2, o_2)$ *defined as* (l, s, o)*, iff* $s_1 = s_2 = s$*,* $l_1 \uplus l_2 = l$*,* $o_1 = l_2 \uplus o$*, and* $o_2 = l_1 \uplus o$*; otherwise it is undefined.*

We use σ to range over these structured heaps. Again following [5], we use abstract commands, A, and abstract boolean tests, b, for our abstract heap model. Note that by encoding each primitive command onto a pair of abstract commands, we can give our language a grainless semantics [20].

Definition 6. *(i) Primitive commands* A *are represented by a subset of* $M \times M$*, satisfying: (1) If* $A(l_1 \uplus l, l_2)$*, then either there exists* l_2' *such that* $A(l_1, l_2')$ *and* $l_2 = l \uplus l_2'$*, or* $\neg\exists l.\ A(l_1, l)$*; and (2) If* $\neg\exists l_2.\ A(l_1 \uplus l, l_2)$*, then* $\neg\exists l_2.\ A(l_1, l_2)$*.*
(ii) Boolean expressions b *are represented by* $M \to \{\mathbf{true}, \mathbf{false}, \mathbf{fault}\}$*, satisfying: if* $b(l_1 \uplus l) = v$*, then either* $b(l_1) = v$ *or* $b(l_1) = \mathbf{fault}$*.*

We present the key rules of the semantics of the abstract programming language in Figure 4. The rest can be found in the extended version [22]. We define a reduction step $\text{Config}_1 \xrightarrow[\lambda]{\mathcal{R}} \text{Config}_2$, as configuration Config_1 makes a reduction step to Config_2 with possible interference $\mathcal{R}$ and label λ. The label indicates whether this is a program action, $\mathbf{p}$, or an environment action, $\mathbf{e}$. Configurations are either **fault** or a pair of a command and a structured heap, (C, σ). We use $\xrightarrow{\mathcal{R}}{}^{*}$ as the transitive and reflex closure of the reduction relation.

[5] If $m_1 \uplus m = m_2 \uplus m$, then $m_1 = m_2$.

[6] The assertions simply ignore the environment.

We alter the syntax of **atomic** to have a postcondition annotation Q, to specify how the state is split between shared and local on exit from the block. In CSL the resource invariant does this job, but we do not have a single resource invariant in this logic. Each of these postconditions must be precise, so there is a unique splitting.[7] Consider the semantics of **atomic** (Figure 4). The non-faulting rule (1) combines the thread's local state with the shared state to create a new local state, $l \uplus s = l_1$, (2) checks the guard holds of this new state, $b(l_1)$, (3) executes the command with no interference on the shared state (**Emp**), (4) splits the resulting local state into a new shared and local state, $l' \uplus s' = l_2$, and (5) finally checks the postcondition Q holds of the shared state s'. As Q is precise, it uniquely specifies the splitting in step (4). There are three more rules for **atomic** (not presented) where the program faults on the evaluation of the body, the evaluation of the guard, or fails to find a splitting to satisfy the postcondition.

Parallel composition is modelled by interleaving, we just present one of the rules. The three remaining rules concern abstract commands and environment transitions. The abstract command A executes correctly, if it runs correctly by accessing only the local state. Otherwise, A faults. Its execution does not affect the shared and environment states. An environment transition can happen anytime and affects only the shared state and the environment state, provided that the shared-state change describes the rely relation, $\mathcal{R}$; the local state is unchanged.

We extend the standard separation logic notion of safety with a guarantee observed by each program action.

Definition 7 (Guarantee). *(1)* $(C, \sigma, \mathcal{R})$ guars_0 $\mathcal{G}$ *always holds; and (2)* $(C, \sigma, \mathcal{R})$ guars_{n+1} $\mathcal{G}$ *iff if* $(C, \sigma) \xrightarrow[\lambda]{\mathcal{R}}$ Config *then there exist* C' σ' *such that* Config $= (C', \sigma')$*;* $(C', \sigma', \mathcal{R})$ guars_n $\mathcal{G}$*; and if* $\lambda = \mathbf{p}$ *then* $(\sigma, \sigma') \in \mathcal{G}$.

Definition 8. $\models C$ **sat** (p, R, G, q) *iff for all* $R' \subseteq R$ *and* $\sigma \vDash_{R'} (p)$*, then (1)* $\forall n. (C, \sigma, [\![R']\!])$ guars_n $[\![G]\!]$*; and (2) if* $(C, \sigma) \xrightarrow{[\![R']\!]}{}^{*} (\mathbf{skip}, \sigma')$ *then* $\sigma' \vDash_{R'} (q)$.

Theorem 2 (Soundness). *If* $\vdash C$ **sat** (p, R, G, q)*, then* $\models C$ **sat** (p, R, G, q)

6 Related Work

Owicki & Gries [16] introduced the concept of non-interference between the proofs of parallel threads. Their method is not compositional and does not permit top-down development of a proof because the final check of interference-freedom may fail rendering the whole development useless.

To address this problem, Jones [11] introduced the compositional rely/guarantee method. In the VDM-style, Jones opted for 'two-state' postconditions; other authors [23,18] have chosen single-state postconditions. Several authors

[7] P is precise iff for every $l \in M$, there exists at most one l_P such that $l_P \vDash_{\mathrm{SL}} P$ and $\exists l'. l_P \uplus l' = l$.

have proved the soundness and relative completeness of rely/guarantee [23,18,7]; Prensa's proof [18] is machine checked by the Isabelle theorem prover. The completeness results are all modulo the introduction of auxiliary variables. Abadi and Lamport [1] have adapted RG to temporal logic and have shown its soundness for safety specifications.

Separation logic [19,15] takes a different approach to interference by forbidding it except in critical regions [10]. An invariant, I, is used to describe the shared state. This is a simple case of our system where the interference specifications (i.e. R and G) are restricted to a very simple relation, $I \rightsquigarrow I$. Brookes has shown concurrent separation logic to be sound [3].

There have been attempts to verify fine-grained concurrent algorithms using both separation logic and rely/guarantee. Vafeiadis *et al.* [21] verify several list algorithms using rely/guarantee. Their proofs require reachability predicates to describe lists and they cannot deal with the disposal of nodes. Parkinson *et al.* [17] verify a non-blocking stack algorithm using concurrent separation logic. Their proof requires a lot of auxiliary state to encode the possible interference. With the logic presented in this paper much of the auxiliary state can be removed, and hence the proof becomes clearer.

Concurrently with our work, Feng, Ferreira and Shao [8] proposed a different combination of rely/guarantee and separation logic, SAGL. Both our approach and SAGL partition memory into shared and private parts. However, in SAGL, every primitive command is assumed to be atomic. Our approach is more flexible and allows one to specify what is atomic; everything else is considered non-atomic. By default, non-atomic commands cannot update shared state, so we only need stability checks when there is an atomic command: in the lock coupling list only at the lock and unlock operations. On the other hand, SAGL must check stability after every single command. Moreover, in SAGL, the rely and guarantee conditions are relations and stability checks are semantic implications. We instead provide convenient syntax for writing down these relations, and reduces the semantic implication into a simple logic implication. This allowed us to automated our logic [6], and hence automatically verify the safety of a collection of fine-grained list algorithms.

SAGL is presented as a logic for assembly code, and is thus hard to apply at different abstraction levels. It does not contain separation logic as a proper subsystem, as it lacks the standard version of the frame rule [19]. This means that it cannot prove the usual separation logic specification of procedures such as `copy_tree` [14]. In contrast, our system subsumes SL [19], as well as the single-resource variant of CSL [15]: hence, the same proofs there (for a single resource) go through directly in our system (for procedures see [22]). Of course, the real interest is the treatment of additional examples, such as lock coupling, that neither separation logic nor rely/guarantee can prove tractably. Our system also includes a rely-guarantee system, which is why we claim to have produced a marriage of the two approaches. It may be possible to extend SAGL to include the frame rule for procedures, but we understand that such extension is by no means obvious.

With this all being said, there are remarkable similarities between our work and SAGL; that they were arrived at independently is perhaps encouraging as to the naturalness of the basic ideas.

7 Conclusion

We have presented a marriage of rely/guarantee with separation logic. We proved soundness with respect to an abstract operational semantics in the style of abstract separation logic [5]. Hence, our proof can be reused with different languages and with different separation logics, e.g. permissions and variables as resource [2]. Our logic allows us to give a clear and simple proof of the lock-coupling list algorithm, which includes memory disposal. Moreover, our logic can be efficiently automated [6].

Acknowledgements. We should like to thank Josh Berdine, Richard Bornat, Cristiano Calcagno, Joey Coleman, Tony Hoare, Cliff Jones, Xinyu Feng, Alan Mycroft, Peter O'Hearn, Uday Reddy, John Reynolds and Hongseok Yang for discussions and feedback on this work. We acknowledge funding from an RAEng/EPSRC fellowship (Parkinson) and a Gates scholarship (Vafeiadis).

References

1. Abadi, M., Lamport, L.: Conjoining specifications. ACM Trans. Prog. Lang. Syst. 17(3), 507–534 (1995)
2. Bornat, R., Calcagno, C., Yang, H.: Variables as resource in separation logic. ENTCS 155, 247–276 (2006)
3. Brookes, S.D.: A semantics for concurrent separation logic. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 16–34. Springer, Heidelberg (2004)
4. Calcagno, C., Gardner, P., Zarfaty, U.: Context logic as modal logic: completeness and parametric inexpressivity. In: POPL, pp. 123–134. ACM Press, New York (2007)
5. Calcagno, C., O'Hearn, P., Yang, H.: Local action and abstract separation logic. LICS (to appear, 2007)
6. Calcagno, C., Parkinson, M., Vafeiadis, V.: Modular safety checking for fine-grained concurrency. In: SAS. LNCS, Springer, Heidelberg (2007)
7. Coleman, J.W., Jones, C.B.: A structural proof of the soundness of rely/guarantee rules. Technical Report CS-TR-987, Newcastle University (October 2006)
8. Feng, X., Ferreira, R., Shao, Z.: On the relationship between concurrent separation logic and assume-guarantee reasoning. In: Proceedings of ESOP (2007)
9. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM 12(10), 576–580 (1969)
10. Hoare, C.A.R.: Towards a theory of parallel programming. Operating Systems Techniques (1971)
11. Jones, C.B.: Specification and design of (parallel) programs. In: IFIP Congress, pp. 321–332 (1983)
12. Jones, C.B.: Wanted: a compositional approach to concurrency, pp. 5–15. Springer, New York (2003)

13. Morgan, C.: The specification statement. ACM Trans. Program. Lang. Syst. 10(3), 403–419 (1988)
14. O'Hearn, P., Reynolds, J., Yang, H.: Local reasoning about programs that alter data structures. In: Proceedings of CSL, pp. 1–19 (2001)
15. O'Hearn, P.W.: Resources, concurrency and local reasoning. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 49–67. Springer, Heidelberg (2004)
16. Owicki, S.S., Gries, D.: An axiomatic proof technique for parallel programs. Acta Informatica 6, 319–340 (1976)
17. Parkinson, M.J., Bornat, R., O'Hearn, P.W.: Modular verification of a non-blocking stack. In: POPL (2007)
18. Prensa Nieto, L.: The rely-guarantee method in Isabelle/HOL. In: Degano, P. (ed.) ESOP 2003. LNCS, vol. 2618, pp. 348–362. Springer, Heidelberg (2003)
19. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS, Washington, DC, USA, pp. 55–74. IEEE Computer Society Press, Los Alamitos (2002)
20. Reynolds, J.C.: Toward a grainless semantics for shared-variable concurrency. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 35–48. Springer, Heidelberg (2004)
21. Vafeiadis, V., Herlihy, M., Hoare, T., Shapiro, M.: Proving correctness of highly-concurrent linearisable objects. In: PPoPP, ACM Press, New York (2006)
22. Vafeiadis, V., Parkinson, M.: A marriage of rely/guarantee and separation logic. Technical Report UCAM-CL-TR-687, University of Cambridge (June 2007), `http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-687.html`
23. Xu, Q., de Roever, W.P., He, J.: The rely-guarantee method for verifying shared variable concurrent programs. Formal Aspects of Computing 9(2), 149–174 (1997)

A Proof Outline: remove

```
remove(e) { local x,y,z,t;
$\{\boxed{\exists A.\ ls(\mathrm{Head}, A, \mathrm{nil}) * s(A)} \land -\infty < \mathrm{e} \land \mathrm{e} < +\infty\}$
  (x,y) = locate(e);
$\left\{\begin{array}{l}\exists uv.\ \boxed{\exists ZAB.\ ls(\mathrm{Head}, A, \mathrm{x}) * L_{\mathrm{tid}}(\mathrm{x}, u, \mathrm{y}) * N(\mathrm{y}, v, Z) * ls(Z, B, \mathrm{nil}) * s(A{\cdot}u{\cdot}v{\cdot}B)} \\ * (\mathrm{x}{+}2{\mapsto}\mathrm{y}) \land u < \mathrm{e} \land \mathrm{e} \le v \land \mathrm{e} < +\infty\end{array}\right\}$
  t = y.value; if (t == e) {
$\left\{\begin{array}{l}\exists u.\ \boxed{\exists ZAB.\ ls(\mathrm{Head}, A, \mathrm{x}) * L_{\mathrm{tid}}(\mathrm{x}, u, \mathrm{y}) * N(\mathrm{y}, \mathrm{e}, Z) * ls(Z, B, \mathrm{nil}) * s(A{\cdot}u{\cdot}\mathrm{e}{\cdot}B)} \\ * (\mathrm{x}{+}2{\mapsto}\mathrm{y}) \land \mathrm{e} < +\infty\end{array}\right\}$
    lock(y);
$\left\{\begin{array}{l}\exists uZ.\ \boxed{\exists AB.\ ls(\mathrm{Head}, A, \mathrm{x}) * L_{\mathrm{tid}}(\mathrm{x}, u, \mathrm{y}) * L_{\mathrm{tid}}(\mathrm{y}, \mathrm{e}, Z) * ls(Z, B, \mathrm{nil}) * s(A{\cdot}u{\cdot}\mathrm{e}{\cdot}B)} \\ * (\mathrm{x}{+}2{\mapsto}\mathrm{y}) * (\mathrm{y}{+}2{\mapsto}Z) \land \mathrm{e} < +\infty\end{array}\right\}$
    z = y.next; x.next = z;
$\left\{\begin{array}{l}\exists u.\ \boxed{\exists AB.\ ls(\mathrm{Head}, A, \mathrm{x}) * L_{\mathrm{tid}}(\mathrm{x}, u, \mathrm{y}) * L_{\mathrm{tid}}(\mathrm{y}, \mathrm{e}, \mathrm{z}) * ls(\mathrm{z}, B, \mathrm{nil}) * s(A{\cdot}u{\cdot}B)} \\ *(\mathrm{x}{+}2{\mapsto}\mathrm{z}) * (\mathrm{y}{+}2{\mapsto}\mathrm{z})\end{array}\right\}$
    unlock(x);
$\{\boxed{\exists A.\ ls(\mathrm{Head}, A, \mathrm{nil}) * s(A)} * L_{\mathrm{tid}}(\mathrm{y}, \mathrm{e}, \mathrm{z}) * (\mathrm{y}{+}2{\mapsto}\mathrm{z})\}$
    dispose(y);
  } else { unlock(x); }
$\{\boxed{\exists A.\ ls(\mathrm{Head}, A, \mathrm{nil}) * s(A)}\}$
}
```

Fair Cooperative Multithreading*

or

Typing Termination in a Higher-Order Concurrent Imperative Language

Gérard Boudol

INRIA, 06902 Sophia Antipolis, France

Abstract. We propose a new operational model for shared variable concurrency, in the context of a concurrent, higher-order imperative language à la ML. In our model the scheduling of threads is cooperative, and a non-terminating process suspends itself on each recursive call. A property to ensure in such a model is fairness, that is, any thread should yield the scheduler after some finite computation. To this end, we follow and adapt the classical method for proving termination in typed formalisms, namely the realizability technique. There is a specific difficulty with higher-order state, which is that one cannot define a realizability interpretation simply by induction on types, because applying a function may have side-effects at types not smaller than the type of the function. Moreover, such higher-order side-effects may give rise to computations that diverge without resorting to explicit recursion. We overcome these difficulties by introducing a type and effect system for our language that enforces a stratification of the memory. The stratification prevents the circularities in the memory that may cause divergence, and allows us to define a realizability interpretation of the types and effects, which we then use to prove the intended termination property.

1 Introduction

This work is concerned with the design of languages for concurrent programming with shared memory. In the recent past, some new applications have emerged, like web servers, network games or large scale databases, that are open to many simultaneous connections or requests, and are therefore inherently massively concurrent. It has been argued that kernel threads usually supported by operating systems are too limited and too inefficient to provide a convenient means for programming such applications, and that a user-level thread facility would provide better support [2,3,6,33]. More generally, it appears that the preemptive discipline for scheduling threads is not very convenient for programming the

* Work partially supported by the CRE FT-R&D no 46136511, and by the ANR-SETI-06-010 grant.

above-mentioned applications, and that an event-driven model, or more generally a cooperative discipline is better suited for this purpose [1,5,6,23].

In the *cooperative* programming model, a thread decides, by means of specific instructions (like yield for instance), when to leave its turn to another concurrent thread, and the scheduling is therefore distributed among the components. In this model, there is no data race, and modular programming can be supported, since using for instance a library function that operates on a shared, mutable data structure (as this is the case of methods attached to objects) does not require rewriting the library code. This is the model that we adopt as the basis for our concurrency semantics.

However, this model also has its drawbacks. A first one is that it does not support true concurrency, that is, it is not well suited to exploit multi-processor architectures. We do not address this issue here (for some work in this direction, see [12,13]). The other flaw of cooperative scheduling is that if the active thread does not cooperate, failing to yield the scheduler, then the model is broken, in the sense that no other component will have a chance to execute. In other words, in cooperative programming, programs *must be cooperative*, or *fair*, that is, they should be guaranteed to either terminate or suspend themselves infinitely often. In particular, this property should be enforced in programming languages of the *reactive* family [10,12,14,21,26,28]. Failing to cooperate may happen for instance if the active thread performs a blocking i/o, or runs into an error, or raises an uncaught exception, or diverges. Some efforts have been done to provide a better support to cooperative scheduling from the operating system [2,3], and to develop asynchronous versions of system services. From the programming language point of view, cooperative programming should better be confined to be used in the framework of a *safe* language, like ML, where a program does not silently fall into an error. However, this is not enough: we have to avoid divergence in some way, while still being able to program non-terminating applications – any server for instance should conceptually have an infinite life duration, and should not be programmed to stop after a while.

In order to ensure fairness in cooperative programming, our proposal is to introduce a specific construct for programming non-terminating processes, the semantics of which is that a looping process suspends itself on each recursive call. We are assuming here that calling ordinary recursive functions – like for instance sorting a list – always terminate. Indeed, non-termination in the evaluation of an expression is usually the symptom of a programming error[1], and it is therefore worth having a distinguished construct for programming intentionally looping processes. The idea of suspensive looping is not, however, enough to ensure fairness in a higher-order imperative programming model à la ML, that we use as a basis here (some other choices are obviously possible). We have to face a technical problem, which is that recursion may be encoded in two ways in a ML-like, or rather, for that matter, a SCHEME-like language. Indeed, it is well-known that one can define a fixpoint combinator in the untyped (call-by-value)

[1] There is a lot of work on techniques for ensuring termination of recursive programs (we refrain from mentioning any paper from the huge literature on this topic), which is not the issue we are addressing here.

λ-calculus. Moreover, as shown long ago by Landin [19], one can implement recursion by means of circular higher-order references (this is indeed the way it is implemented), like in

$$\begin{aligned} F &= (\mathsf{let}\ f = (\mathsf{ref}\ \lambda x x)\ \mathsf{in}\ f := \lambda x((\,!\,f)x)\ ;\ !\,f) \\ &\simeq \mathsf{rec}\, f(x)(fx) \end{aligned} \tag{1}$$

where we use ML's notations $(\mathsf{ref}\ V)$ for creating a reference with initial value V, and $!\,u$ for reading the value of the reference u. The well-known method to recover from the first difficulty, disallowing the ability to derive fixpoint combinators, is to use a *type system*, but this is not enough to ensure termination of non-recursive programs in an imperative and functional language: in a simple type system, the expression F above has type $(\tau \rightarrow \tau)$ (it can indeed be written in OCaml for instance), but it diverges when applied to any value. As far as we can see, nothing has ever been proposed to ensure termination in a higher-order *imperative* (and concurrent) language, thus disallowing implicit recursion via the store. In this work we show that we can use a type and *effect* system [20] for this purpose. This is our main technical contribution.

Among the arguments used to show termination in typed higher-order formalisms, the realizability method is perhaps the best known, and certainly the most widely applicable. The realizability technique consists in defining, by induction on the structure of types[2], an interpretation of types as sets of expressions, so that

1. the interpretation of a type only contains expressions enjoying the intended computational property (e.g. weak or strong normalizability);
2. typing is *sound*: a typed expression belongs to the interpretation of its type, or *realizes* its type.

The main ingredient in the definition of such an interpretation of types is that an expression M realizes a functional type $(\tau \rightarrow \sigma)$ if and only if its application (MN) to any argument N realizing τ is an expression that realizes σ. A realizability interpretation is therefore a special case of a "logical relation" [22,25]. Such a realizability interpretation was first introduced by Kleene for intuitionistic arithmetic [17], though not for the purpose of proving termination. The technique was then used by Tait in [30], under the name of "convertibility" (with no reference to Kleene's notion of realizability), to show (weak) normalizability in the simply typed λ-calculus, and subsequently by Girard (see [16]) with his "candidats de réductibilité", and by Tait again [31] (who related it to Kleene's work) to show strong normalizability in higher-order typed λ-calculi. As a matter of fact, this technique seems to apply to most type theories – see the textbooks [4,16,18]. It has also been used for (functional fragments of) higher-order process calculi, and most notably the π-calculus [27,35].

However, as far as I can see, the realizability technique has not been previously used for higher-order imperative (and concurrent) languages: the work that is technically the closest to ours, and which was our main source of inspiration,

[2] A more elaborate definition has to be used in the case of recursive types, see [8].

is the one by Pitts and Stark [24], who introduced logical relations to provide means to prove observational equivalence of programs (not to prove termination), but their language is restricted to offer only storable values of basic types (this has been slightly relaxed in [7]). The program of Example (1) shows the main difficulty in attempting to define a realizability interpretation for higher-order imperative languages: to define the interpretation of a type τ, one should have previously defined the interpretation of the types of values stored in the memory that an expression of type τ may manipulate, but these types have no reason to be strictly smaller than τ. As another example, unrelated to divergence, one may imagine a function, say from lists of integers to integers, that reads from the memory (or import from a module) a second-order function like map, and uses it for its own computations.

To preclude the circularities in the memory that may cause recursion-free divergence, our type and effect system stratifies the memory into *regions*, in such a way that functional values stored in a given region may only have a latent effect, such as reading a reference, in strictly "lower" regions, thus rejecting (1) for instance. This stratification is also the key to defining a realizability interpretation, by a sophisticated induction over types and effects. We introduce such a realizability interpretation, for which our type and effect system is sound. From this we conclude that any typable program is fair.

The paper is organized as follows: we first define the syntax and operational semantics of our core language, introducing a "yield-and-loop" construct for programming non-terminating applications, and a new way of managing threads over an ML-like language. Then we define our type and effect system, where the main novelty is the region typing context, introducing a stratification into the memory. Next we show our type safety result. To this end we introduce a realizability interpretation of the types and effects, and show that the type system is sound with respect to this interpretation. We then briefly conclude.

Note. For lack of space, the proofs are omitted. They can be found in the full version of the paper, available from the author's web page.

2 Syntax

Our core concurrent programming language is an ML-like language, that is a call-by-value λ-calculus extended with imperative constructs for creating, reading and updating references in the memory, and enriched with concurrent programming constructs. The latter include a thread-spawning construct, and a "yield-and-loop" value $\nabla y M$, to be run by applying it to a void value (). This is our main linguistic novelty. This is similar to a recursive function $\mathsf{rec}\, y()M$, but we wish to stress the fact that the semantics is quite different. An expression $(\nabla y M())$ represents a recursive process which *yields the scheduler*, while unfolding a copy of M (where y is recursively bound to $\nabla y M$) to be performed when the scheduler resumes it. This construct is useful to build non-terminating processes, which should not hold the scheduler forever. This is the only form of explicit recursion that we shall consider here. In a more realistic language, we would not only include ordinary recursive functions $\mathsf{rec}\, f(x)M$, but also consider synchronization constructs, like the ones of reactive programming for instance [10,12,21,26,28].

We assume given an infinite set $\mathcal{R}eg$ of *region names* (or simply regions), ranged over by ρ. We also assume given an infinite set $\mathcal{V}ar$ of *variables*, ranged over by x, y, $z\ldots$, and an infinite set $\mathcal{L}oc$ of abstract *memory locations*. We let u, $v\ldots$ range over $\mathcal{L}oc$. A *reference* is a pair (u,ρ), that we will always denote by u_ρ, of a memory location to which a region is assigned. The set $\mathcal{R}ef$ of references is therefore $\mathcal{L}oc \times \mathcal{R}eg$. We shall denote $\mathcal{L}oc \times \{\rho\}$ as $\mathcal{L}oc_\rho$. The syntax of our core language is as follows:

$$\begin{array}{rcll} M,\ N\ldots & ::= & V \ \mid\ (MN) & \textit{expressions}\\ & \mid & (\mathsf{ref}_\rho M)\ \mid\ (\mathbf{!}\,M)\ \mid\ (M := N) & \\ & \mid & (\mathsf{thread}\, M) & \\ V,\ W\ldots & ::= & x\ \mid\ \lambda x M\ \mid\ ()\ \mid\ \nabla y M\ \mid\ u_\rho & \textit{values} \end{array}$$

We require reference creation $(\mathsf{ref}_\rho M)$ to occur in an explicitly given region ρ, although in a type and effect inference approach (see [29]) this could perhaps be inferred. We denote by $\mathcal{V}al$ the set of values. As usual, the variable x is bound in $\lambda x M$, and similarly the variable y is bound in $\nabla y M$. We denote by $\{x \mapsto V\}M$ the capture-avoiding substitution of the value V for the free occurrences of the variable x in M. We shall consider expressions up to α-conversion, that is up to the renaming of bound variables. We use the standard notations for $(\lambda x MN)$, namely $(\mathsf{let}\ x = N\ \mathsf{in}\ M)$, and $(N\,\mathbf{;}\,M)$ when x is not free in M.

The operational semantics will be defined as a transition relation between configurations, that involve in particular the current expression to evaluate, and a pool of threads waiting for execution. In order to get a *fair* scheduling strategy, we split this pool of threads into two parts, or more precisely two *turns*, that are multisets of expressions. Then a *configuration* is a tuple $C = (\delta, M, T, S)$, where

- δ is the memory,
- M is the currently evaluated expression (the active thread),
- T is the multiset of threads in the *current turn* of execution,
- S is the multiset of threads waiting for the *next turn*.

The memory δ in a configuration is a mapping from a finite subset $\mathsf{dom}(\delta)$ of $\mathcal{R}ef$ to the set $\mathcal{V}al$ of values, such that to each memory address is assigned only one region, that is

$$u_{\rho_0} \in \mathsf{dom}(\delta)\ \&\ u_{\rho_1} \in \mathsf{dom}(\delta)\ \Rightarrow\ \rho_0 = \rho_1$$

We shall suppose given a function fresh from the set $\mathcal{P}_f(\mathcal{R}ef)$ of finite subsets of $\mathcal{R}ef$ to $\mathcal{L}oc$ such that $\mathsf{fresh}(R)_\rho \notin R$ for all ρ.

As regards multisets, our notations are as follows. Given a set X, a *multiset* over X is a mapping E from X to the set $\mathbb{N}$ of non-negative integers, indicating the *multiplicity* $E(x)$ of an element. We denote by $\mathsf{0}$ the empty multiset, such that $\mathsf{0}(x) = 0$ for any x, and by x the singleton multiset such that $x(y) = (\mathsf{if}\ y = x\ \mathsf{then}\ 1\ \mathsf{else}\ 0)$. Multiset union $E + E'$ is given by $(E+E')(x) = E(x) + E'(x)$. In the following we only consider multisets of expressions, ranged over by $S, T\ldots$

$$\begin{array}{rcll}
(\delta, \mathbf{E}[(\lambda x M V)]) & \xrightarrow[\emptyset]{} & (\delta, \mathbf{E}[\{x \mapsto V\} M], 0, 0) & \\
(\delta, \mathbf{E}[(\mathsf{ref}_\rho V)]) & \xrightarrow[\{\rho\}]{} & (\delta \cup \{u_\rho \mapsto V\}, \mathbf{E}[u_\rho], 0, 0) & u = \mathsf{fresh}(\mathsf{dom}(\delta)) \\
(\delta, \mathbf{E}[(!\, u_\rho)]) & \xrightarrow[\{\rho\}]{} & (\delta, \mathbf{E}[V], 0, 0) & V = \delta(u_\rho) \\
(\delta, \mathbf{E}[(u_\rho := V)]) & \xrightarrow[\{\rho\}]{} & (\delta[u_\rho := V], \mathbf{E}[()], 0, 0) & \\
(\delta, \mathbf{E}[(\mathsf{thread}\, M)]) & \xrightarrow[\emptyset]{} & (\delta, \mathbf{E}[()], M, 0) & \\
(\delta, \mathbf{E}[(\nabla y M V)]) & \xrightarrow[\emptyset]{} & (\delta, (), 0, \mathbf{E}[\{y \mapsto \nabla y M\} M]) &
\end{array}$$

Fig. 1. Operational Semantics (Sequential)

The operational semantics consists in reducing the active expression in a configuration into a value, and this, as usual, means reducing a "redex" (reducible expression) inside an *evaluation context* (see [34]). Then the last syntactic ingredient we have to define is the one of evaluation contexts. This is given by the following grammar:

$$\begin{array}{llr}
\mathbf{E} ::= [] \mid \mathbf{E}[\mathbf{F}] & & \textit{evaluation contexts} \\
\mathbf{F} := ([]\, N) \mid (V[]) \mid (\mathsf{ref}_\rho []) \mid (!\, []) & & \textit{frames} \\
\quad \mid ([] := N) \mid (V := []) \mid ([] \backslash \rho) & &
\end{array}$$

As usual, we shall only consider for execution well-formed configurations. Roughly speaking, a configuration (δ, M, T, S) is *well-formed*, written $(\delta, M, T, S)\,\mathrm{wf}$, if and only if all the references occurring in the configuration are bound to a value in the memory (we omit the obvious formal definition). We are now ready to define the operational semantics of our language.

3 Operational Semantics

In order to define the transition relation between configurations, we first define the sequential evaluation of expressions. This is given by an annotated transition relation

$$(\delta, M) \xrightarrow[e]{} (\delta', M', T, S)$$

where T and S are the multisets (which actually are either empty or a singleton) of threads spawned at this step, for execution in the current and next turn respectively, and e is the *effect* at this step. As usual, (imperative) effects record the regions in which an expression may operate, either by creating, reading or udpating a reference. In what follows it will not be necessary to distinguish different kinds of effects, and therefore an effect is simply a (finite) *set of regions*. We denote by $\mathcal{E}\!f\!f$ the set of effects, that is $\mathcal{E}\!f\!f = \mathcal{P}_f(\mathcal{R}eg)$.

The sequential part of the operational semantics is given in Figure 1. This part is quite standard, except as regards the looping construct $\nabla y M$. An expression $\mathbf{E}[(\nabla y M V)]$ instantly terminates, returning $()$, while spawning as a thread the unfolding $\{y \mapsto \nabla y M\} M$ of the loop, in its evaluation context $\mathbf{E}$. One should notice that the thread $\mathbf{E}[\{y \mapsto \nabla y M\} M]$ created by means of the looping construct is delayed to be executed at the *next* turn, whereas with the construct

$$\frac{(\delta, M) \xrightarrow[e]{} (\delta', M', T', S')}{(\delta, M, T, S) \xrightarrow[e]{} (\delta', M', T + T', S + S')} \text{ (Exec)}$$

$$\frac{}{(\delta, V, N + T, S) \xrightarrow[\emptyset]{} (\delta, N, T, S)} \text{ (Sched 1)} \qquad \frac{}{(\delta, V, \mathsf{0}, N + T) \xrightarrow[\emptyset]{} (\delta, N, T, \mathsf{0})} \text{ (Sched 2)}$$

Fig. 2. Operational Semantics (Concurrent)

$(\mathsf{thread}\, M)$ the new thread M is to be executed during the *current* turn. Then in order to execute immediately (and recursively) some task M, one should rather use the following construct:

$$\mu y M =_{\text{def}} \{y \mapsto \nabla y M\} M$$

For instance one can define $(\mathsf{loop}\, M) = \mu y (M \,\mathsf{;}\, (y()))$, which repeatedly starts executing M until termination, and then resumes at the next turn. To code a service $(\mathsf{repeat}\, M)$ that has to execute some task M at every turn, like continuously processing requests to a server for instance, one would write – using standard conventions for saving some parentheses:

$$(\mathsf{repeat}\, M) =_{\text{def}} \mu y. (\mathsf{thread}\, y()) \,\mathsf{;}\, M$$

Anticipating on the concurrency semantics, we can describe the behaviour of this expression as follows: it spawns a new thread $N = (\nabla y ((\mathsf{thread}\, (y())) \,\mathsf{;}\, M)())$ in the current turn of execution, and starts M. Whenever the thread N comes to be executed, during the current turn, a thread performing $(\mathsf{repeat}\, M)$ is spawned for execution at the next turn. Notice that the ability of spawning threads for execution in the current turn is required for writing such a program.

Our concurrency semantics, which, together with the "yield-and-loop" construct, is the main novelty of this work, is defined in Figure 2, which we now comment. We see from the rule (Exec) that the active expression keeps executing, possibly spawning new threads, till termination. When this expression is terminated, a scheduling operation occurs: if there is some thread waiting for execution in the current turn, the value returned by the previously active thread is discarded, and a thread currently waiting is non-deterministically elected for becoming active, as stated by rule (Sched 1). Otherwise, by the rule (Sched 2), one chooses to execute a thread that was waiting for the next turn, if any, and simultaneously the other "next-turn" threads all become "current-turn" ones. If there is no waiting thread, the execution stops. (It should be obvious that reduction preserves the well-formedness of configurations.) One should notice that the termination of the active thread may be temporary. This is the case when the thread is actually performing a looping operation. Indeed, if we define

$$\mathsf{yield} = (\nabla y()())$$

then the execution of a thread $\mathbf{E}[\mathsf{yield}]$ stops, and will resume executing $\mathbf{E}[()]$ at the next turn. That is, we have

$$(\delta, \mathbf{E}[\mathsf{yield}], T, S) \rightarrow (\delta, (), T, S + \mathbf{E}[()])$$

To conclude this section we introduce some notations. We shall denote by $\rightarrow^a$ the transition relation between configurations that only involves the active expression, that is, the transition relation defined as $\rightarrow$, but without using the rules (SCHED 1) and (SCHED 2). Similarly, we denote by $\rightarrow^c$ the transitions that occur in the current turn of execution. This is defined as $\rightarrow$, but without using (SCHED 2). Then the sequences of $\rightarrow$ transitions can be decomposed into a sequence of $\rightarrow^c$ transitions, then possibly an application of the (SCHED 2) rule, then again a sequence of $\rightarrow^c$ transitions, and so on. Following the terminology of synchronous or reactive programming [12,21,28], a maximal sequence of $\rightarrow^c$ transitions may be called an *instant*. Then the property we wish to ensure is that all the instants in the execution of a program are finite. More precisely, we define:

Definition (Reactivity) 3.1. *A configuration is* reactive *if all the maximal sequences of $\rightarrow^c$ transitions originating from that configuration are finite, and end up with a configuration of the form $(\delta, V, 0, S)$.*

Notice that, in particular, a reactive configuration is guaranteed not to spawn infinitely many threads for execution in the current turn.

For the following technical developments, it will be convenient to introduce some more notations. First, we define $\overset{*}{\underset{e}{\rightarrow}}{}^a$ as follows:

$$\frac{}{C \overset{*}{\underset{e}{\rightarrow}}{}^a C} \qquad \frac{C \underset{e}{\rightarrow}{}^a C'' \overset{*}{\underset{e'}{\rightarrow}}{}^a C'}{C \overset{*}{\underset{e \cup e'}{\longrightarrow}}{}^a C'}$$

The relation $\overset{*}{\underset{e}{\rightarrow}}{}^c$ is defined in the same way. Next, in order to show our termination property, we need to take into account the possible interleavings of threads in the current turn. More precisely, we observe that in the execution of (δ, M, T, S), the thread (hereditarily) created by M may be run starting with a memory that results from executing some threads in T (and possibly threads hereditarily created by threads in T). Then, given a set $\mathcal{M}$ of memories, we define a transition relation $\rightarrow^{c,\mathcal{M}}$ which is given as $\rightarrow^c$, except that in the case where a scheduling occurs, the new thread may be started in the context of any memory from $\mathcal{M}$:

$$\frac{(\delta, M) \underset{e}{\rightarrow} (\delta', M', T', S')}{(\delta, M, T, S) \underset{e}{\rightarrow}{}^{c,\mathcal{M}} (\delta', M', T + T', S + S')} \qquad \frac{\delta' \in \mathcal{M} \quad (\delta', N, T, S) \text{ wf}}{(\delta, V, N + T, S) \underset{\emptyset}{\rightarrow}{}^{c,\mathcal{M}} (\delta', N, T, S)}$$

We shall also use the relation $\overset{*}{\underset{e}{\rightarrow}}{}^{c,\mathcal{M}}$, defined in the same way as $\overset{*}{\underset{e}{\rightarrow}}{}^c$.

Definition ($\mathcal{M}$-Convergence) 3.2. *Given a set $\mathcal{M}$ of memories, a closed expression M* converges *w.r.t. $\mathcal{M}$, in notation $M{\Downarrow}_{\mathcal{M}}$ if and only if, for all $\delta \in \mathcal{M}$, if $(\delta, M, 0, 0)$ is well-formed, then there is no infinite sequence of $\rightarrow^{c,\mathcal{M}}$ transitions from this configuration, and each maximal such sequence ends up with a configuration of the form $(\delta', V, 0, S)$.*

4 The Type and Effect System

The types are

$$\tau,\ \sigma,\ \theta\ldots \in \mathcal{T}\!ype\ ::=\ \mathbb{1}\ \mid\ \theta\,\mathsf{ref}_\rho\ \mid\ (\tau \xrightarrow{e} \sigma)$$

The type $\mathbb{1}$ is also denoted unit (and sometimes improperly void). As usual, in the functional types $(\tau \xrightarrow{e} \sigma)$ we record the latent effect e, that is the effect a value of this type may have when applied to an argument. We define the size $|\tau|$ and the set $\mathsf{reg}(\tau)$ of regions that occur in a latent effect in τ as follows:

$$\begin{array}{rclcrcl} |\mathbb{1}| &=& 0 & \qquad & \mathsf{reg}(\mathbb{1}) &=& \emptyset \\ |\theta\,\mathsf{ref}_\rho| &=& 1 + |\theta| & & \mathsf{reg}(\theta\,\mathsf{ref}_\rho) &=& \mathsf{reg}(\theta) \\ |\tau \xrightarrow{e} \sigma| &=& 1 + |\tau| + |\sigma| & & \mathsf{reg}(\tau \xrightarrow{e} \sigma) &=& \mathsf{reg}(\tau) \cup e \cup \mathsf{reg}(\sigma) \end{array}$$

We shall say that a type τ is *pure* if it does not mention any imperative effect, that is $\mathsf{reg}(\tau) = \emptyset$.

In order to rule out from the memory the circularities that may cause divergence in computations, we assign a type to each region, in such a way that the region cannot be reached by using a value stored in that region. This is achieved, as in dependent type systems [4], by introducing the notion of a well-formed type with respect to a type assignment to regions. A *region typing context* Δ is a *sequence* $\rho_1 : \theta_1, \ldots, \rho_n : \theta_n$ of assignments of types to regions. We denote by $\mathsf{dom}(\Delta)$ the set of regions where Δ is defined. Then we define by simultaneous induction two predicates $\Delta \vdash$, for "the context Δ is well-formed", and $\Delta \vdash \tau$, for "the type τ is well-formed in the context of Δ", as follows:

$$\frac{}{\emptyset \vdash} \qquad\qquad \frac{\Delta \vdash \theta}{\Delta, \rho : \theta \vdash}\ \rho \notin \mathsf{dom}(\Delta)$$

$$\frac{\Delta \vdash}{\Delta \vdash \mathbb{1}} \qquad \frac{\Delta \vdash \quad \Delta(\rho) = \theta}{\Delta \vdash \theta\,\mathsf{ref}_\rho} \qquad \frac{\Delta \vdash \tau \quad \Delta \vdash \sigma \quad e \subseteq \mathsf{dom}(\Delta)}{\Delta \vdash (\tau \xrightarrow{e} \sigma)}$$

For any well-formed region typing context Δ, we denote by $\mathcal{ET}(\Delta)$ the set of pairs (e, τ) of an effect and a type such that $e \subseteq \mathsf{dom}(\Delta)$ and $\Delta \vdash \tau$. One may observe that if $\rho_1 : \theta_1, \ldots, \rho_n : \theta_n \vdash$ then $i \neq j \Rightarrow \rho_i \neq \rho_j$. Moreover, it is easy to see that

$$\Delta \vdash \tau\ \Rightarrow\ \mathsf{reg}(\tau) \subseteq \mathsf{dom}(\Delta)$$

and therefore

$$\Delta \vdash \theta\,\mathsf{ref}_\rho\ \Rightarrow\ \rho \notin \mathsf{reg}(\theta) \tag{2}$$

The important clause in the definition of well-formedness is the last one: to be well-formed in the context of Δ, the type $(\tau \xrightarrow{e} \sigma)$ of a function with side-effects must be such that all the regions involved in the latent effect e are already recorded in Δ. (This is vacuously true if there are no such regions, and in particular if the functional type is pure. Indeed, if τ is pure, we have $\Delta \vdash \tau$ for any well-formed Δ.) This is the way we will avoid "dangerous" circularities in the memory. For instance, if $\Delta \vdash (\tau \xrightarrow{e} \sigma)$ and $\rho \in e$, then the type $(\tau \xrightarrow{e} \sigma)\,\mathsf{ref}_\rho$ is not well-formed in the context of Δ, thanks to the remark (2) above.

The judgements of the type and effect system for our source language have the form $\Delta; \Gamma \vdash M : e, \tau$, where Γ is a typing context in the usual sense, that is

$$\frac{\Delta \vdash \Gamma \quad \Gamma(x) = \tau}{\Delta; \Gamma \vdash x : \emptyset, \tau} \qquad \frac{\Delta; \Gamma, x : \tau \vdash M : e, \sigma \quad \Delta \vdash (\tau \xrightarrow{e} \sigma)}{\Delta; \Gamma \vdash \lambda x M : \emptyset, (\tau \xrightarrow{e} \sigma)}$$

$$\frac{\Delta; \Gamma \vdash M : e, (\tau \xrightarrow{e''} \sigma) \quad \Delta; \Gamma \vdash N : e', \tau}{\Delta; \Gamma \vdash (MN) : e \cup e' \cup e'', \sigma} \qquad \frac{\Delta; \Gamma \vdash M : e, \theta \quad \Delta(\rho) = \theta}{\Delta; \Gamma \vdash (\mathsf{ref}_\rho M) : \{\rho\} \cup e, \theta\, \mathsf{ref}_\rho}$$

$$\frac{\Delta; \Gamma \vdash M : e, \theta\, \mathsf{ref}_\rho \quad \Delta; \Gamma \vdash N : e', \theta}{\Delta; \Gamma \vdash (M := N) : \{\rho\} \cup e \cup e', \mathbb{1}} \qquad \frac{\Delta; \Gamma \vdash M : e, \theta\, \mathsf{ref}_\rho}{\Delta; \Gamma \vdash (\mathbf{!}\, M) : \{\rho\} \cup e, \theta}$$

$$\frac{\Delta \vdash \Gamma}{\Delta; \Gamma \vdash () : \emptyset, \mathbb{1}} \qquad \frac{\Delta; \Gamma \vdash M : e, \tau}{\Delta; \Gamma \vdash (\mathsf{thread}\, M) : e, \mathbb{1}}$$

$$\frac{\Delta; \Gamma, y : (\mathbb{1} \xrightarrow{e} \mathbb{1}) \vdash M : e, \mathbb{1}}{\Delta; \Gamma \vdash \nabla y M : \emptyset, (\mathbb{1} \xrightarrow{e} \mathbb{1})} \qquad \frac{\Delta \vdash \tau \quad \Delta; \Gamma \vdash M : e, \sigma}{\Delta; \Gamma, x : \tau \vdash M : e, \sigma}\ x \notin \mathsf{dom}(\Gamma)$$

Fig. 3. Type and Effect System

a mapping from a finite set $\mathsf{dom}(\Gamma)$ of variables to types. We omit this context when it has an empty domain, writing $\Delta; \vdash M : e, \tau$ in this case. We denote by $\Gamma, x : \tau$ the typing context which is defined as Γ, except for x, to which is assigned the type τ. We extend the well-formedness of types predicate to typing contexts, as follows:

$$\Delta \vdash \Gamma \quad \Leftrightarrow_{\mathrm{def}} \quad \Delta \vdash \ \& \ \forall x \in \mathsf{dom}(\Gamma).\ \Delta \vdash \Gamma(x)$$

The rules of the type and effect system are given in Figure 3. The typing rules are standard, except for the fact that we check well-formedness with respect to the region typing context Δ. Then our type system conservatively extends the usual simple type system for pure functions. One can see that some expressions that read above their type are typable, like $((\mathbf{!}\, u_\rho)N)$ where $\Delta(\rho) = (\tau \xrightarrow{\emptyset} \sigma)$ and N is of type τ (a more interesting example, using constructs for manipulating lists, was suggested in the Introduction). As a matter of fact, it is always safe to read functions of a pure type from the memory.

In order to state our Type Safety property, we have to extend the typing to configurations and, first, to memories:

$$\Delta; \Gamma \vdash \delta \quad \Leftrightarrow_{\mathrm{def}} \quad \forall u_\rho.\ u_\rho \in \mathsf{dom}(\delta) \Rightarrow \begin{cases} \rho \in \mathsf{dom}(\Delta)\ \& \\ \Delta; \Gamma \vdash \delta(u_\rho) : \emptyset, \Delta(\rho) \end{cases}$$

The typing judgements are also extended to multisets of expressions, as follows:

$$\frac{}{\Delta; \Gamma \vdash \mathsf{0} : \emptyset} \qquad \frac{\Delta; \Gamma \vdash M : e, \tau \quad \Delta; \Gamma \vdash T : e'}{\Delta; \Gamma \vdash M + T : e \cup e'}$$

Then we define

$$\Delta;\Gamma \vdash (\delta, M, T, S) : e \ \Leftrightarrow_{\mathrm{def}} \ \begin{cases} \Delta;\Gamma \vdash \delta \ \& \\ \exists e_0 \subseteq e.\ \exists\tau.\ \Delta;\Gamma \vdash M : e_0, \tau \ \& \\ \exists e_1 \subseteq e.\ \exists e_2.\ \Delta;\Gamma \vdash T : e_1 \ \& \ \Delta;\Gamma \vdash S : e_2 \end{cases}$$

5 The Termination Property

In this section we define the *realizability predicate* which, given a well-formed region typing context Δ, states that an expression M realizes the effect e and the type τ in the context of Δ, in notation $\Delta \models M : e, \tau$. This is defined by induction on e and τ, with respect to a well-founded ordering that we now introduce. First, for each region typing Δ and type τ, we define the set $\mathsf{Reg}_\Delta(\tau)$, which intuitively is the set of regions in $\mathsf{dom}(\Delta)$ that are involved in a proof that τ is well-formed, in the case where $\Delta \vdash \tau$. This includes in particular the regions of $\mathsf{Reg}_\Delta(\theta)$ whenever τ is a functional type, and θ is the type assigned in Δ to a region that occurs in the latent effect of τ. Then, overloading the notation, we also define $\mathsf{Reg}_\Delta(R)$ for $R \subseteq \mathcal{R}eg$. The definition of $\mathsf{Reg}_\Delta(\tau)$ and $\mathsf{Reg}_\Delta(R)$ is by simultaneous induction on (the length of) Δ. For any given Δ, $\mathsf{Reg}_\Delta(\tau)$ is defined by induction on τ, in a uniform way:

$$\begin{aligned} \mathsf{Reg}_\Delta(\mathbb{1}) &= \emptyset \\ \mathsf{Reg}_\Delta(\theta\,\mathsf{ref}_\rho) &= \mathsf{Reg}_\Delta(\theta) \\ \mathsf{Reg}_\Delta(\tau \xrightarrow{e} \sigma) &= \mathsf{Reg}_\Delta(\tau) \cup \mathsf{Reg}_\Delta(\sigma) \cup \mathsf{Reg}_\Delta(e) \end{aligned}$$

Then $\mathsf{Reg}_\Delta(R)$ is given by:

$$\mathsf{Reg}_\emptyset(R) = \emptyset$$

$$\mathsf{Reg}_{\Delta,\rho:\theta}(R) = \begin{cases} \{\rho\} \cup \mathsf{Reg}_\Delta(R - \{\rho\}) \cup \mathsf{Reg}_\Delta(\theta) & \text{if } \rho \in R \\ \mathsf{Reg}_\Delta(R) & \text{otherwise} \end{cases}$$

It is easy to see that $\mathsf{reg}(\tau) \subseteq \mathsf{Reg}_\Delta(\tau) \subseteq \mathsf{dom}(\Delta)$ if $\Delta \vdash \tau$, and that $R \subseteq \mathsf{Reg}_\Delta(R)$ if $R \subseteq \mathsf{dom}(\Delta)$. Moreover, if τ is pure, then $\mathsf{Reg}_\Delta(\tau) = \emptyset$. The following is an equally easy but crucial remark:

Lemma 5.1. *If $\Delta \vdash$ and $\theta = \Delta(\rho)$, where $\rho \in \mathsf{dom}(\Delta)$, then $\mathsf{Reg}_\Delta(\theta) \subset \mathsf{Reg}_\Delta(\{\rho\})$.*

(The proof, by induction on Δ, is trivial, since $\Delta, \rho : \theta \vdash$ implies $\rho \notin \mathsf{dom}(\Delta)$ and $\Delta \vdash \theta$.) The realizability interpretation is defined by induction on a strict ordering on the pairs (e, τ), namely the lexicographic ordering on $(\mathsf{Reg}_\Delta(e) \cup \mathsf{Reg}_\Delta(\tau), |\tau|)$. More precisely, we define:

Definition (Effect and Type Strict Ordering) 5.2. *Let Δ be a well-formed region typing context. The relation $\prec_\Delta$ on $\mathcal{ET}(\Delta)$ is defined as follows: $(e, \tau) \prec_\Delta (e', \tau')$ if and only if*

(i) $\mathsf{Reg}_\Delta(e) \cup \mathsf{Reg}_\Delta(\tau) \subset \mathsf{Reg}_\Delta(e') \cup \mathsf{Reg}_\Delta(\tau')$, *or*

(ii) $\mathsf{Reg}_\Delta(e) \cup \mathsf{Reg}_\Delta(\tau) = \mathsf{Reg}_\Delta(e') \cup \mathsf{Reg}_\Delta(\tau')$ *and* $|\tau| < |\tau'|$.

We notice two facts about this ordering:

1. for pure types, this ordering is the usual one, that is $(\emptyset, \tau) \prec_\Delta (\emptyset, \sigma)$ if and only if $|\tau| < |\sigma|$;
2. the pure types are always smaller than impure ones, that is $(\emptyset, \tau) \prec_\Delta (\emptyset, \sigma)$ if $\mathsf{reg}(\tau) = \emptyset \neq \mathsf{reg}(\sigma)$.

The strict ordering $\prec_\Delta$ is *well-founded*, that is, there is no infinite sequence $(e_n, \tau_n)_{n\in\mathbb{N}}$ in $\mathcal{ET}(\Delta)$ such that $(e_{n+1}, \tau_{n+1}) \prec_\Delta (e_n, \tau_n)$ for all n. Notice that, by the lemma above, we have in particular $(\emptyset, \theta) \prec_\Delta (e, \tau)$ if $\theta \in \Delta(e)$.

Our realizability interpretation states that if an expression M realizes a type, then in particular it converges in the context of suitable memories. As explained in the Introduction, realizability has to be defined for the types of values that M may read or modify in the memory, and this is what we mean by "suitable." The portion of the memory that has to be "suitable" may be restricted to the regions where M may have a side-effect (as approximated by the type and effect system). In the following definition we let, for $X \subseteq \mathcal{R}eg$:

$$\Delta \models \delta \upharpoonright X \quad \Leftrightarrow_{\text{def}} \quad \forall \rho \in X \cap \mathsf{dom}(\Delta).\ \forall u_\rho \in \mathsf{dom}(\delta).\ \Delta \models \delta(u_\rho) : \emptyset, \Delta(\rho)$$

Clearly, $\Delta \models \delta \upharpoonright X$ is vacuously true if $X = \emptyset$.

Definition (Realizability) 5.3. *The closed expression M* realizes e, τ *in the context of Δ, in notation $\Delta \models M : e, \tau$, if and only if the following holds, where we let $\mathcal{M} = \{\, \delta \mid \Delta \models \delta \upharpoonright e \,\}$:*
(i) $(e, \tau) \in \mathcal{ET}(\Delta)$;
(ii) $M{\Downarrow}_{\mathcal{M}}$;
(iii) $\delta \in \mathcal{M}$ & $(\delta, M, 0, 0) \xrightarrow[e']{*}{}^{c,\mathcal{M}} (\delta', M', T, S) \ \Rightarrow\ \delta' \in \mathcal{M}$;
(iv) *if* $\delta \in \mathcal{M}$ *and* $(\delta, M, 0, 0) \xrightarrow[e']{*}{}^{a} (\delta', V, T, 0)$ *then*
  (a) *if* $\tau = \mathbb{1}$ *then* $V = ()$,
  (b) *if* $\tau = \theta\,\mathsf{ref}_\rho$ *then* $V \in \mathcal{L}oc_\rho$,
  (c) *if* $\tau = (\theta \xrightarrow{e''} \sigma)$ *then* $\forall W.\ \Delta \models W : \emptyset, \theta \ \Rightarrow\ \Delta \models (VW) : e'', \sigma$.

This is extended to open expressions as follows: if $\mathsf{fv}(M) \subseteq \mathsf{dom}(\Gamma)$ *where* $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$ *then* $\Delta; \Gamma \models M : e, \tau$ *if and only if*

$$\forall i\, \forall V_i.\ \Delta \models V_i : \tau_i \Rightarrow \Delta \models \{x_1 \mapsto V_1, \ldots, x_n \mapsto V_n\} M : e, \tau$$

Notice that the hypothesis of the item (iv) of the definition means in particular that reducing M does not end up performing a call to a recursive process $\nabla y N$. This definition is well-founded[3]. Indeed, with the statement $\Delta \models \delta \upharpoonright e$ the definition of $\Delta \models M : e, \tau$ calls for $\Delta \models V : \emptyset, \theta$ where $\theta = \Delta(\rho)$ for some ρ in e (if there is any such region), and we have seen that $(\emptyset, \theta) \prec_\Delta (e, \tau)$ in this case. If $\tau = (\theta \xrightarrow{e''} \sigma)$, the definition calls for $\Delta \models W : \emptyset, \theta$ and $\Delta \models N : e'', \sigma$. It is

[3] In [15], a notion of "imperative realizability" is defined for an ML-like language, with some restrictions to preclude aliasing, but it is not completely clear to me that this definition is well-founded. A similar remark seems to apply to [9].

clear that, in this case, $(\emptyset, \theta) \prec_\Delta (e, \theta \xrightarrow{e''} \sigma)$ since $\mathsf{Reg}_\Delta(\theta) \subseteq \mathsf{Reg}_\Delta(\theta \xrightarrow{e''} \sigma)$ and $|\theta| < |\theta \xrightarrow{e''} \sigma|$. Moreover, since $\mathsf{Reg}_\Delta(e'') \subseteq \mathsf{Reg}_\Delta(\theta \xrightarrow{e''} \sigma)$, it is obvious that $(e'', \sigma) \prec_\Delta (e, \theta \xrightarrow{e''} \sigma)$.

We can now state our main result, namely a Type Safety theorem, which improves upon the standard statement:

Theorem (Type Safety) 5.4. *If (δ, M, T, S) is a closed, well-formed typable configuration, that is $\Delta; \vdash (\delta, M, T, S) : e$ for some region typing Δ and effect e, then (δ, M, T, S) is reactive. Moreover, any configuration reachable from (δ, M, T, S) is reactive.*

For lack of space, the proof is omitted (it can be found in the full version of the paper). To show this result, we establish the *soundness* of the type system with respect to the realizability interpretation, namely that if an expression has effect e and type τ in some context, then in the same context it realizes e and τ (see [4,18], and also [22], where soundness is called "the Basic Lemma").

6 Conclusion

We have proposed a way of ensuring fairness in a cooperative, concurrent, higher-order imperative language, by introducing a specific construct for programming non-terminating processes. Our main technical contribution consists in designing a type and effect system for our language, that supports an extension of the classical realizability technique to show our termination property, namely fairness.

Our study was limited to a very simple core language, and clearly it should be extended to more realistic ones. The synchronization constructs of synchronous, or reactive programming for instance [10,12,14,21,26,28] should be added. We believe this should not cause any difficulty. Indeed, the problem with termination in a concurrent higher-order imperative language is in the interplay between functions and store, and between recursion and thread creation. In the full version of the paper, we show a way of dealing with ordinary recursive functions: we prove a result that is stronger than the one we presented here, namely that if a typable configuration does not perform an infinite number of calls to (ordinary) recursive functions, then it is reactive, and only leads to reactive configurations. This is achieved using the same realizability technique, with a continuity argument (*cf.* [24]) to deal with recursive functions. In the full version of the paper, we distinguish shared and unshared regions, and we restrict the threads to have visible effects only in shared regions (relative to which some compiler or hardware optimizations should be disallowed, like with "volatile" variables). To compensate for this restriction, we include into the language the *effect masking* construct of [20], that is $(\mathsf{local}\ \rho\ \mathsf{in}\ M)$, binding the region ρ to be used only by M. This construct allows typing an expression using only local state as "pure". Such pure expressions may be executed as processes, in preemptive mode, and possibly on different processors. The correctness of the typing of effect masking is shown as part of a standard subject reduction property, using a new method that differs from the one of [32].

Another topic that deserves to be investigated is whether the restriction imposed by our stratification of the memory is acceptable in practice. We believe that the restriction we have on the storable functional values is not too severe (in particular, any pure function can be stored), but obviously our design for the type system needs to be extended, and experimented on real applications, in order to assess more firmly this belief. We notice also that our approach does not seem to preclude the use of cyclic data structures. In OCaml for instance, one may define cyclic lists like – using standard notations for the list constructors:

$$(\mathsf{let\ rec}\ x = \mathsf{cons}(1, x)\ \mathsf{in}\ x)$$

Such a value, which is a list of integers, does not show any effect, and therefore it should be possible to extend our language and type and effect system to deal with such circular data structures.

Finally it would be interesting to see whether showing termination in a concurrent, higher-order imperative language may have other applications than the one which motivated our work (*cf.* [11]), and whether our realizabity interpretation could be generalized to logical relations (and to richer notions of type), in order to prove program equivalences for instance. This is left for further investigations.

References

1. Adya, A., Howell, J., Theimer, M., Bolosky, W.J., Douceur, H.R.: Cooperative task management without manual stack management or, Event-driven programming is not the opposite of threaded programming, Usenix ATC (2002)
2. Anderson, T.E., Bershad, B.N., Lazowska, E.D., Levy, H.M.: Scheduler activations: effective kernel support for the user-level management of parallelism. ACM Trans. on Computer Systems 10(1), 53–79 (1992)
3. Banga, G., Druschel, P., Mogul, J.C.: Better operating sytem features for faster network servers. ACM SIGMETRICS Performance Evaluation Review 26(3), 23–30 (1998)
4. Barendregt, H.: Lambda Calculi with Types. In: Abramsky, S., Gabbay, D.M., Maibaum, T.S.E. (eds.) Handbook of Logic in Computer Science, pp. 117–309. Oxford University Press, Oxford (1992)
5. von Berhen, R., Condit, J., Brewer, E.: Why events are a bad idea (for highconcurrency servers). In: Proceedings of HotOS IX (2003)
6. von Berhen, R., Condit, J., Zhou, F., Necula, G.C., Brewer, E.: Capriccio: scalable threads for Internet services. In: SOSP'03 (2003)
7. Benton, N., Leperchey, B.: Relational reasoning in a nominal semantics for storage. In: Urzyczyn, P. (ed.) TLCA 2005. LNCS, vol. 3461, pp. 86–101. Springer, Heidelberg (2005)
8. Birkedal, L., Harper, R.: Relational interpretation of recursive types in an operational setting. Information and Computation 155(1-2), 3–63 (1999)
9. Bohr, N., Birkedal, L.: Relational reasoning for recursive types and references. In: Kobayashi, N. (ed.) APLAS 2006. LNCS, vol. 4279, pp. 79–96. Springer, Heidelberg (2006)
10. Boudol, G.: ULM, a core programming model for global computing. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 234–248. Springer, Heidelberg (2004)
11. Boudol, G.: On typing information flow. In: Van Hung, D., Wirsing, M. (eds.) ICTAC 2005. LNCS, vol. 3722, pp. 366–380. Springer, Heidelberg (2005)

12. Boussinot, F.: FairThreads: mixing cooperative and preemptive threads in C. Concurrency and Computation: Practice and Experience 18, 445–469 (2006)
13. Dabrowski, F., Boussinot, F.: Cooperative threads and preemptive computations. In: Proceeding of TV'06, Workshop on Multithreading in Hardware and Software: Formal Approaches to Design and Verification, FLoC'06 (2006)
14. Epardaud, S.: Mobile reactive programming in ULM. In: Proc. of the Fifth ACM SIGPLAN Workshop on Scheme and Functional Programming, pp. 87–98. ACM Press, New York (2004)
15. Filliâtre, J.-C.: Verification of non-functional programs using interpretations in type theory. J. Functional Programming 13(4), 709–745 (2003)
16. Girard, J.-Y., Lafont, Y., Taylor, P.: Proofs and Types. Cambridge Tracts in Theoretical Computer Science, vol. 7. Cambridge University Press, Cambridge (1989)
17. Kleene, S.C.: On the interpretation of intuitionistic number theory. J. of Symbolic Logic 10, 109–124 (1945)
18. Krivine, J.-L.: Lambda-Calcul: Types et Modèles, Masson, Paris (1990). English translation Lambda-Calculus, Types and Models, Ellis Horwood (1993)
19. Landin, P.J.: The mechanical evaluation of expressions. Computer Journal 6, 308–320 (1964)
20. Lucassen, J.M., Gifford, D.K.: Polymorphic effect systems. In: POPL'88, pp. 47–57 (1988)
21. Mandel, L., Pouzet, M.: ReactiveML, a reactive extension to ML. In: PPDP'05, pp. 82–93 (2005)
22. Mitchell, J.C.: Foundations for Programming Languages. MIT Press, Cambridge (1996)
23. Ousterhout, J.: Why threads are a bad idea (for most purposes), presentation given at the 1996 Usenix ATC (1996)
24. Pitts, A., Stark, I.: Operational reasoning for functions with local state. In: Gordon, A., Pitts, A. (eds.) Higher-Order Operational Techniques in Semantics, pp. 227–273. Publication of the Newton Institute, Cambridge Univ. Press (1998)
25. Plotkin, G.: Lambda-definability and logical relations. Memo SAI-RM-4, University of Edinburgh (1973)
26. Pucella, R.: Reactive programming in Standard ML. In: IEEE Intern. Conf. on Computer Languages, pp. 48–57. IEEE Computer Society Press, Los Alamitos (1998)
27. Sangiorgi, D.: Termination of processes. Math. Struct. in Comp. Science 16, 1–39 (2006)
28. Serrano, M., Boussinot, F., Serpette, B.: Scheme fair threads. In: PPDP'04, pp. 203–214 (2004)
29. Talpin, J.-P., Jouvelot, P.: The type and effect discipline. Information and Computation 111, 245–296 (1994)
30. Tait, W.: Intensional interpretations of functionals of finite type I. J. of Symbolic Logic 32, 198–212 (1967)
31. Tait, W.: A realizability interpretation of the theory of species. Logic Colloquium, Lecture Notes in Mathematics, vol. 453, pp. 240–251 (1975)
32. Tofte, M., Talpin, J.-P.: Region-based memory management. Information and Computation 132(2), 109–176 (1997)
33. Welsh, M., Culler, D., Brewer, E.: SEDA: an architecture for well-conditioned, scalable internet services. In: SOSP'01, pp. 230–243 (2001)
34. Wright, A., Felleisen, M.: A syntactic approach to type soundness. Information and Computation 115(1), 38–94 (1994)
35. Yoshida, N., Berger, M., Honda, K.: Strong normalisation in the π-calculus. Information and Computation 191(2), 145–202 (2004)

Precise Fixpoint-Based Analysis of Programs with Thread-Creation and Procedures

Peter Lammich and Markus Müller-Olm

Institut für Informatik, Fachbereich Mathematik und Informatik
Westfälische Wilhelms-Universität Münster
peter.lammich@uni-muenster.de, mmo@math.uni-muenster.de

Abstract. We present a fixpoint-based algorithm for context-sensitive interprocedural kill/gen-analysis of programs with thread creation. Our algorithm is precise up to abstraction of synchronization common in this line of research; it can handle forward as well as backward problems. We exploit a structural property of kill/gen-problems that allows us to analyze the influence of environment actions independently from the local transfer of data flow information. While this idea has been used for programs with parbegin/parend blocks before in work of Knoop/Steffen/Vollmer and Seidl/Steffen, considerable refinement and modification is needed to extend it to thread creation, in particular for backward problems. Our algorithm computes annotations for all program points in time depending linearly on the program size, thus being faster than a recently proposed automata based algorithm by Bouajjani et. al..

1 Introduction

As programming languages with explicit support for parallelism, such as Java, have become popular, the interest in analysis of parallel programs has increased in recent years. Most papers on precise analysis, such as [5,13,10,9,3,4], use parbegin/parend blocks or their interprocedural counterpart, parallel procedure calls, as a model for parallelism. However, this is not adequate for analyzing languages like Java, because in presence of procedures or methods the thread-creation primitives used in such languages cannot be simulated by parbegin/parend [1]. This paper presents an efficient, fixpoint-based algorithm for precise kill/gen-analysis of programs with both thread-creation and parallel calls.

Due to known undecidability and complexity results efficient and precise analyses can only be expected for program models that ignore certain aspects of behavior. As common in this line of research (compare e.g. [13,10,9,3,4,1]) we consider flow- and context-sensitive analysis of a program model without synchronization. Note that by a well-known result of Ramalingam [12], context- and synchronization-sensitive analysis is undecidable. We focus on kill/gen problems, a practically relevant class of dataflow problems that comprises the well-known bitvector problems, e.g. live variables, available expressions, etc. Note that only slightly more powerful analyses, like copy constants or truly live variables are intractable or even undecidable (depending on the atomicity of assignments) for parallel programs [10,9].

Extending previous work [5], Seidl and Steffen proposed an efficient, fixpoint-based algorithm for precise kill/gen-analysis of programs with parallel procedure calls [13]. Adopting their idea of using a separate analysis of possible interference, we construct an algorithm that treats thread creation in addition to parallel call. This extension requires considerable modification. In particular, possible interference has a different nature in presence of thread creation because a thread can survive the procedure that creates it. Also backwards analysis is inherently different from forward analysis in presence of thread creation. As our algorithm handles both thread creation and parallel procedure calls it strictly generalizes Seidl and Steffen's algorithm from [13]. It also treats backwards kill/gen problems for arbitrary programs, while [13] assumes that every forward reachable program point is also backwards reachable.

In [1], an automata based approach to reachability analysis of a slightly stronger program model than ours is presented. In order to compute bitvector analysis information for multiple program points, which is often useful in the context of program optimization, this automata based algorithm must be iterated for each program point, each iteration needing at least linear time in the program size. In contrast, our algorithm computes the analysis information for *all* program points in linear time. Moreover, our algorithm can compute with whole bitvectors, exploiting efficiently implementable bitvector operations, whereas the automata based algorithm must be iterated for each bit. To the best of our knowledge, there has been no precise interprocedural analysis of programs with thread creation that computes information for all program points in linear time.

In a preliminary and less general version of this paper [8], we already covered forward analysis for programs without parallel calls.

This paper is organized as follows: After defining flow graphs and their operational semantics in Section 2, we define the class of kill/gen analysis problems and their *MOP-solution*, using the operational semantics as a reference point (Section 3). We then develop a fixpoint-based characterization of the MOP-solution amenable to algorithmic treatment for both forward and backward problems (Sections 4 and 5), thereby relying on information about the *possible interference*, whose computation is deferred to Section 6. We generalize our treatment to parallel procedure calls in Section 7. Section 8 indicates how to construct a linear-time algorithm from our results and discusses future research.

2 Parallel Flow Graphs

A parallel flowgraph $(P, (G_p)_{p\in P})$ consists of a finite set P of procedure names, with $\mathsf{main} \in P$. For each procedure $p \in P$, there is a directed, edge annotated finite graph $G_p = (N_p, E_p, \mathsf{e}_p, \mathsf{r}_p)$ where N_p is the set of control nodes of procedure p and $E_p \subseteq N_p \times \mathcal{A} \times N_p$ is the set of edges that are annotated with base, call or spawn statements: $\mathcal{A} := \{\mathsf{base}\ b \mid b \in \mathcal{B}\} \cup \{\mathsf{call}\ p \mid p \in P\} \cup \{\mathsf{spawn}\ p \mid p \in P\}$.

The set $\mathcal{B}$ of base edge annotations is not specified further for the rest of this paper. Each procedure $p \in P$ has an entry node $\mathsf{e}_p \in N_p$ and a return node $\mathsf{r}_p \in N_p$. As usual we assume that the nodes of the procedures are disjoint, i.e. $N_p \cap N_{p'} = \emptyset$ for $p \neq p'$ and define $N = \bigcup_{p \in P} N_p$ and $E = \bigcup_{p \in P} E_p$.

We use $\mathcal{M}(X)$ to denote the set of multisets of elements from X. The empty multiset is $\emptyset$, $\{a\}$ is the multiset containing a once and $A \uplus B$ is multiset union.

We describe the operational semantics of a flowgraph by a labeled transition system $\overset{\cdot}{\longrightarrow} \subseteq \mathsf{Conf} \times \mathcal{L} \times \mathsf{Conf}$ over configurations $\mathsf{Conf} := \mathcal{M}(N^*)$ and labels $\mathcal{L} := E \cup \{\mathsf{ret}\}$. A configuration consists of the stacks of all threads running in parallel. A stack is modeled as a list of control nodes, the first element being the current control node at the top of the stack. We use $[]$ for the empty list, $[e]$ for the list containing just e and write $r_1 r_2$ for the concatenation of r_1 and r_2. Execution starts with the initial configuration, $\{[\mathsf{e}_{\mathsf{main}}]\}$. Each transition is labeled with the corresponding edge in the flowgraph or with ret for the return from a procedure. We use an *interleaving semantics*, nondeterministically picking the thread that performs the next transition among all available threads. Thus we define $\overset{\cdot}{\longrightarrow}$ by the following rules:

[base]	$(\{[u]r\} \uplus c) \xrightarrow{e} (\{[v]r\} \uplus c)$	for edges $e = (u, \mathsf{base}\ a, v) \in E$
[call]	$(\{[u]r\} \uplus c) \xrightarrow{e} (\{[\mathsf{e}_q][v]r\} \uplus c)$	for edges $e = (u, \mathsf{call}\ q, v) \in E$
[ret]	$(\{[\mathsf{r}_q]r\} \uplus c) \xrightarrow{\mathsf{ret}} (\{r\} \uplus c)$	for procedures $q \in P$
[spawn]	$(\{[u]r\} \uplus c) \xrightarrow{e} (\{[v]r\} \uplus \{[\mathsf{e}_q]\} \uplus c)$	for edges $e = (u, \mathsf{spawn}\ q, v) \in E$

We extend $\xrightarrow{e}$ to finite sequences $w \in \mathcal{L}^*$ in the obvious way. For technical reasons, we assume that every edge $e \in E$ of the flowgraph is dynamically reachable, i.e. there is a path of the form $\{[\mathsf{e}_{\mathsf{main}}]\} \xrightarrow{w[e]} \_$ (we use _ as wildcard, i.e. an anonymous, existentially quantified variable). This assumption is harmless, as unreachable edges can be determined by a simple analysis and then removed which does not affect the analysis information we are interested in.

3 Dataflow Analysis

Dataflow analysis provides a generic lattice-based framework for constructing program analyses. A specific dataflow analysis is described by a tuple $(L, \sqsubseteq, f)$ where $(L, \sqsubseteq)$ is a complete lattice representing analysis information and $f : \mathcal{L} \to (L \overset{\mathsf{mon}}{\to} L)$ maps a transition label e to a monotonic function f_e that describes how a transition labeled e transforms analysis information. We assume that only base-transitions have transformers other than the identity and extend transformers to finite paths by $f_{e_1 \ldots e_n} := f_{e_n} \circ \ldots \circ f_{e_1}$.

In this paper we consider kill/gen-analyses, i.e. we require $(L, \sqsubseteq)$ to be distributive and the transformers to have the form $f_e(x) = (x \sqcap \mathsf{kill}_e) \sqcup \mathsf{gen}_e$ for some $\mathsf{kill}_e, \mathsf{gen}_e \in L$. Note that all transformers of this form are monotonic and that the set of these transformers is closed under composition of functions. In order to allow effective fixpoint computation, we assume that $(L, \sqsubseteq)$ has finite height. As $(L, \sqsubseteq)$ is distributive, this implies that the meet operation distributes over arbitrary joins, i.e. $(\bigsqcup M) \sqcap x = \bigsqcup\{m \sqcap x \mid m \in M\}$ for all $x \in L$ and $M \subseteq L$. Thus, all transformers are positively disjunctive which is important for precise abstract interpretation.

Kill/gen-analyses comprise classic analyses like determination of live variables, available expressions or potentially uninitialized variables.

Depending on the analysis problem, one distinguishes forward and backward dataflow analyses. The analysis information for forward analysis is an abstraction of the program executions that reach a certain control node, while the backward analysis information concerns executions that leave the control node.

The *forward analysis problem* is to calculate, for each control node $u \in N$, the least upper bound of the transformers of all paths reaching a configuration at control node u applied to an initial value x_0 describing the analysis information valid at program start. A configuration $c \in \mathsf{Conf}$ is *at* control node $u \in N$ (we write $\mathsf{at}_u(c)$), iff it contains a stack with top node u. We call the solution of the forward analysis problem MOP^F and define

$$\mathsf{MOP}^\mathsf{F}[u] := \alpha^\mathsf{F}(\mathsf{Reach}[u])$$

where $\mathsf{Reach}[u] := \{w \mid \exists c : \{[\mathsf{e}_\mathsf{main}]\} \xrightarrow{w} c \wedge \mathsf{at}_u(c)\}$ is the set of paths reaching u and $\alpha^\mathsf{F}(W) := \bigsqcup\{f_w(x_0) \mid w \in W\}$ is the abstraction function from concrete sets of reaching paths to the abstract analysis information we are interested in.[1]

The *backward analysis problem* is to calculate, for each control node $u \in N$, the least upper bound of the transformers of all reversed paths leaving reachable configurations at control node u, applied to the least element of L, $\bot_L$. We call the solution of the backward analysis problem MOP^B and define

$$\mathsf{MOP}^\mathsf{B}[u] := \alpha^\mathsf{B}(\mathsf{Leave}[u])$$

where $\mathsf{Leave}[u] := \{w \mid \exists c : \{[\mathsf{e}_\mathsf{main}]\} \xrightarrow{*} c \xrightarrow{w} \_ \wedge \mathsf{at}_u(c)\}$ is the set of paths leaving u, $\alpha^\mathsf{B}(W) := \bigsqcup\{f_{w^R}(\bot_L) \mid w \in W\}$ is the corresponding abstraction function and w^R denotes the word w in reverse order, e.g. $f_{(e_1 \ldots e_n)^R} = f_{e_1} \circ \ldots \circ f_{e_n}$.

Note that we do not use an initial value in the definition of backward analysis, because we have no notion of termination that would give us a point where to apply the initial value. This model is adequate for interactive programs that read and write data while running.

4 Forward Analysis

Abstract interpretation [2,11] is a standard and convenient tool for constructing fixpoint-based analysis algorithms and arguing about their soundness and precision. Thus, a natural idea would be to compute the MOP^F-solution as an abstract interpretation of a system of equations or inequations (*constraint system*) that characterizes the sets $\mathsf{Reach}[u]$.

[1] MOP originally stands for meet over all paths. We use the dual lattice here, but stick to the term MOP for historical reasons.

Unfortunately, it follows from results in [1] that no such constraint system exists in presence of thread creation and procedures (using the natural operators "concatenation" and "interleaving" from [13]). In order to avoid this problem, we derive an alternative characterization of the MOP-solution as the join of two values, each of which can be captured by abstract interpretation of appropriate constraint systems. In order to justify this alternative characterization, we argue at the level of program paths. That is, we perform part of the abstraction already on the path level. More specifically, we classify the transitions of a reaching path into *directly reaching transitions* and *interfering transitions* and show that these transitions are quite independent. We then show how to obtain the MOP-solution from the set of *directly reaching paths* (consisting of directly reaching transitions only) and the *possible interference* (the set of interfering transitions on reaching paths), and how to characterize these sets as constraint systems. The idea of calculating the MOP-solution using possible interference is used already by [13] in a setting with parallel procedure calls. However, while in [13] this idea is used just in order to reduce the size of the constraint system, in our case it is essential in order to obtain a constraint-based characterization at all, due to results of [1] mentioned above.

The classification of transitions is illustrated by Fig. 1. The vertical lines symbolize the executions of single threads, horizontal arrows are thread creation. The path depicted in this figure reaches the control node u in one thread. The directly reaching transitions are marked with thick lines. The other transitions are interfering transitions, which are executed concurrently with the directly reaching transitions, so that the whole path is some interleaving of directly reaching and interfering transitions.

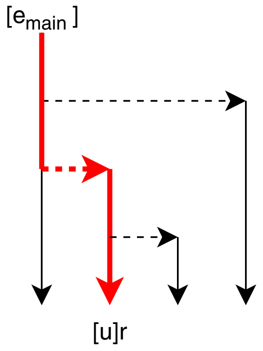


Fig. 1. Directly reaching and interfering transitions

A key observation is that due to the lack of synchronization each potentially interfering transition e can take place at the very end of some path reaching u; thus, the information at u cannot be stronger than gen_e. In order to account for this, we use the least upper bound of all this gen_e-values (see Theorem 2 and the definition of α^{PI} below). This already covers the effect of interfering transitions completely: the kill_e-parts have no strengthening effect for the information at u, because the directly reaching path reaches u without executing any of the interfering transitions.

In order to formalize these ideas, we distinguish directly reaching from interfering transitions in the operational semantics by marking one stack of a configuration as executing directly reaching transitions. Transitions of unmarked stacks are interfering ones. If the marked stack executes a spawn, the marker can either stay with this stack or be transferred to the newly created thread. In Fig. 1 this corresponds to pushing the marker along the thick lines.

In the actual formalization, we mark a single control node in a stack instead of the stack as a whole. This is mainly in order to allow us a more smooth generalization to the case of parallel procedure calls (Section 7). In a procedure call from a marked node we either move the marker up the stack to the level of the newly created procedure or retain it at the level of the calling procedure. Note that we can move the marker from the initial configuration $\{[\mathsf{e}_{\mathsf{main}}]\}$ to any reachable control node u by just using transitions on control nodes above the marker. These transitions formalize the directly reaching transitions. The notation for a node that may be marked is u^m, with $m \in \{\circ, \bullet\}$ and $u \in N$ where u° means that the node is not marked, and $u^\bullet$ means that the node is marked. We now define the following rule templates, instantiated for different values of x below:

$$\begin{array}{lll}
[\text{base}] & (\{[u^m]r\} \uplus c) \xrightarrow{e}_x (\{[v^m]r\} \uplus c) & e = (u, \mathsf{base}\ a, v) \in E \\
[\text{call}] & (\{[u^m]r\} \uplus c) \xrightarrow{e}_x (\{[\mathsf{e}_q^\circ][v^m]r\} \uplus c) & e = (u, \mathsf{call}\ q, v) \in E \\
[\text{ret}] & (\{[\mathsf{r}_q^\circ]r\} \uplus c) \xrightarrow{\mathsf{ret}}_x (\{r\} \uplus c) & q \in P \\
[\text{spawn}] & (\{[u^m]r\} \uplus c) \xrightarrow{e}_x (\{[v^m]r\} \uplus \{[\mathsf{e}_q^\circ]\} \uplus c) & e = (u, \mathsf{spawn}\ q, v) \in E
\end{array}$$

Using these templates, we define the transition relations $\overset{\cdot}{\longrightarrow}_{\mathsf{m}}$ and $\overset{\cdot}{\longrightarrow}_{\mathsf{i}}$. The relation $\overset{\cdot}{\longrightarrow}_{\mathsf{m}}$ is defined by adding the additional side condition that some node in $[u^m]r$ must be marked to the rules [base], [call] and [spawn]. The [ret]-rule gets the additional condition that some node in r must be marked (in particular, r must not be empty). The relation $\overset{\cdot}{\longrightarrow}_{\mathsf{i}}$ is defined by adding the condition that no node in $[u^m]r$ or r respectively must be marked.

Intuitively, $\overset{\cdot}{\longrightarrow}_{\mathsf{m}}$ describes transitions on marked stacks only, but cannot change the position of the marker; $\overset{\cdot}{\longrightarrow}_{\mathsf{i}}$ captures interfering transitions. To be able to push the marker to called procedures or spawned threads, we define the transition relation $\overset{\cdot}{\longrightarrow}_{\mathsf{p}}$ by the following rules:

$$\begin{array}{lll}
[\text{call.push}] & (\{[u^\bullet]r\} \uplus c) \xrightarrow{e}_{\mathsf{p}} (\{[\mathsf{e}_q^\bullet][v^\circ]r\} \uplus c) & e = (u, \mathsf{call}\ q, v) \in E \\
[\text{spawn.push}] & (\{[u^\bullet]r\} \uplus c) \xrightarrow{e}_{\mathsf{p}} (\{[v^\circ]r\} \uplus \{[\mathsf{e}_q^\bullet]\} \uplus c) & e = (u, \mathsf{spawn}\ q, v) \in E
\end{array}$$

According to the ideas described above, we get:

Lemma 1. *Given a reaching path* $\{[\mathsf{e}_{\mathsf{main}}]\} \xrightarrow{w} \{[u]r\} \uplus c$, *there are paths* w_1, w_2 *with* $w \in w_1 \otimes w_2$, *such that* $\exists \hat{c} : \{[\mathsf{e}_{\mathsf{main}}^\bullet]\} \xrightarrow{w_1}_{\mathsf{mp}} \{[u^\bullet]r\} \uplus \hat{c} \xrightarrow{w_2}_{\mathsf{i}} \{[u^\bullet]r\} \uplus c$.

Here, $w_1 \otimes w_2$ denotes the set of all interleavings of the finite sequences w_1 and w_2 and $\overset{\cdot}{\longrightarrow}_{\mathsf{mp}} := \overset{\cdot}{\longrightarrow}_{\mathsf{m}} \cup \overset{\cdot}{\longrightarrow}_{\mathsf{p}}$ executes the directly reaching transitions, resulting in the configuration $\{[u^\bullet]r\} \uplus \hat{c}$. The interfering transitions in w_2 operate on the threads from $\hat{c}$. These threads are either freshly spawned and hence in their initial configuration with just the thread's entry point on the stack, or they have been left by a transition according to rule [spawn.push] and hence are at the target node of the spawn edge and may have some return nodes on the stack.

Now we define the set $\mathsf{R}^{\mathsf{op}}[u]$ of directly reaching paths to u as

$$\mathsf{R}^{\mathsf{op}}[u] := \{w \mid \exists r, c : \{[\mathsf{e}^{\bullet}_{\mathsf{main}}]\} \xrightarrow{w}_{\mathsf{mp}} \{[u^{\bullet}]r\} \uplus c\}$$

and the possible interference at u as

$$\mathsf{PI}^{\mathsf{op}}[u] := \{e \mid \exists r, c, w : \{[\mathsf{e}^{\bullet}_{\mathsf{main}}]\} \xrightarrow{*}_{\mathsf{mp}} \{[u^{\bullet}]r\} \uplus c \xrightarrow{w[e]}_{\mathsf{i}} \_\} \,.$$

The following theorem characterizes the $\mathsf{MOP}^{\mathsf{F}}$-solution based on R^{op} and $\mathsf{PI}^{\mathsf{op}}$:

Theorem 2. $\mathsf{MOP}^{\mathsf{F}}[u] = \alpha^{\mathsf{F}}(\mathsf{R}^{\mathsf{op}}[u]) \sqcup \alpha^{\mathsf{PI}}(\mathsf{PI}^{\mathsf{op}}[u])$

Here, $\alpha^{\mathsf{PI}}(E) := \bigsqcup\{\mathsf{gen}_e \mid e \in E\}$ abstracts sets of edges to the least upper bound of their gen_e-values.

Proof. For the $\sqsubseteq$-direction, we fix a reaching path $w \in \mathsf{Reach}[u]$ and show that its abstraction $f_w(x_0)$ is smaller than the right hand side. Using Lemma 1 we split w into the directly reaching path w_1 and the interfering transitions w_2, such that $w \in w_1 \otimes w_2$. Because we use kill/gen-analysis over distributive lattices, we have the approximation $f_w(x_0) \sqsubseteq f_{w_1}(x_0) \sqcup \bigsqcup\{\mathsf{gen}_e \mid e \in w_2\}$ [13]. Obviously, these two parts are smaller than $\alpha^{\mathsf{F}}(\mathsf{R}^{\mathsf{op}}[u])$ and $\alpha^{\mathsf{PI}}(\mathsf{PI}^{\mathsf{op}}[u])$ respectively, and thus the proposition follows.

For the $\sqsupseteq$-direction, we first observe that any directly reaching path is also a reaching path, hence $\mathsf{MOP}^{\mathsf{F}}[u] \sqsupseteq \alpha^{\mathsf{F}}(\mathsf{R}^{\mathsf{op}}[u])$. Moreover, for each transition $e \in \mathsf{PI}^{\mathsf{op}}[u]$ a path $w[e] \in \mathsf{Reach}[u]$ can be constructed. Its abstraction $(f_w(x_0) \sqcap \mathsf{kill}_e) \sqcup \mathsf{gen}_e$ is obviously greater than gen_e. Thus, also $\mathsf{MOP}^{\mathsf{F}}[u] \sqsupseteq \alpha^{\mathsf{PI}}(\mathsf{PI}^{\mathsf{op}}[u])$. Altogether the proposition follows. □

Constraint systems. In order to compute the right hand side of the equation in Theorem 2 by abstract interpretation, we characterize the directly reaching paths and the possible interference as the least solutions of constraint systems. We will focus on the directly reaching paths here. The constraints for the possible interference are developed in Section 6, because we can reuse results from backward analysis for their characterization. In order to precisely treat procedures, we use a well-known technique from interprocedural program analysis, that first characterizes so called *same-level paths* and then uses them to assemble the directly reaching paths. A same-level path starts and ends at the same stack-level, and never returns below this stack level. We are interested in the same-level paths starting at the entry node of a procedure and ending at some node u of this procedure. We define the set of these paths as $\mathsf{S}^{\mathsf{op}}[u] := \{w \mid \exists c : \{[\mathsf{e}^{\bullet}_p]\} \xrightarrow{w}_{\mathsf{m}} \{[u^{\bullet}]\} \uplus c\}$ for $u \in N_p$. It is straightforward to show that $\mathsf{lfp}(\mathsf{S}) = \mathsf{S}^{\mathsf{op}}$ for the least solution $\mathsf{lfp}(\mathsf{S})$ of the constraint system S over variables $\mathsf{S}[u] \in \mathcal{P}(\mathcal{L}^*)$, $u \in N$ with the following constraints:

$$\begin{array}{lll}
[\text{init}] & \mathsf{S}[\mathsf{e}_q] \supseteq \{\varepsilon\} & \text{for } q \in P \\
[\text{base}] & \mathsf{S}[v] \supseteq \mathsf{S}[u]; e & \text{for } e = (u, \mathsf{base}\ \_, v) \in E \\
[\text{call}] & \mathsf{S}[v] \supseteq \mathsf{S}[u]; e; \mathsf{S}[\mathsf{r}_q]; \mathsf{ret} & \text{for } e = (u, \mathsf{call}\ q, v) \in E \\
[\text{spawn}] & \mathsf{S}[v] \supseteq \mathsf{S}[u]; e & \text{for } e = (u, \mathsf{spawn}\ q, v) \in E
\end{array}$$

The operator ; is list concatenation lifted to sets. The directly reaching paths are characterized by the constraint system R over variables $\mathsf{R}[u] \in \mathcal{P}(\mathcal{L}^*)$, $u \in N$ with the following constraints:

$$\begin{array}{lrll}
\text{[init]} & \mathsf{R}[\mathsf{e}_{\mathsf{main}}] \supseteq \{\varepsilon\} & \\
\text{[reach]} & \mathsf{R}[u] \supseteq \mathsf{R}[\mathsf{e}_p]; \mathsf{S}^{\mathsf{op}}[u] & \text{for } u \in N_p \\
\text{[callp]} & \mathsf{R}[\mathsf{e}_q] \supseteq \mathsf{R}[u]; e & \text{for } e = (u, \mathsf{call}\ q, \_) \in E \\
\text{[spawnp]} & \mathsf{R}[\mathsf{e}_q] \supseteq \mathsf{R}[u]; e & \text{for } e = (u, \mathsf{spawn}\ q, \_) \in E
\end{array}$$

Intuitively, the constraint [reach] corresponds to the transitions that can be performed by the $\xrightarrow{\cdot}_{\mathsf{m}}$ part of $\xrightarrow{\cdot}_{\mathsf{mp}}$, and the [callp]- and [spawnp]-constraints correspond to the $\xrightarrow{\cdot}_{\mathsf{p}}$ part. It is again straightforward to show $\mathsf{lfp}(\mathsf{R}) = \mathsf{R}^{\mathsf{op}}$.

Using standard techniques of abstract interpretation [2,11], we can construct an abstract version $\mathsf{R}^\#$ of R over the domain $(L, \sqsubseteq)$ using an abstract version $\mathsf{S}^\#$ of S over the domain $(L \overset{\mathsf{mon}}{\rightarrow} L, \sqsubseteq)$ and show:

Theorem 3. $\mathsf{lfp}(\mathsf{R}^\#) = \alpha^{\mathsf{F}}(\mathsf{lfp}(\mathsf{R}))$.

5 Backward Analysis

For backward analysis, we consider the paths leaving u. Recall that these are the paths starting at a reachable configuration of the form $\{[u]r\} \uplus c$. Such a path is an interleaving of a path from $[u]r$ and transitions originating from c. The latter ones are covered by the possible interference $\mathsf{PI}^{\mathsf{op}}[u]$. It turns out that in order to come to grips with this interleaving we can use a similar technique as for forward analysis. We define the *directly leaving paths* as

$$\mathsf{L}^{\mathsf{op}}[u] := \{w \mid \exists r, c : \{[\mathsf{e}_{\mathsf{main}}]\} \xrightarrow{*} \{[u]r\} \uplus c \wedge \{[u]r\} \xrightarrow{w} \_\}$$

and show the following characterization:

Theorem 4. $\mathsf{MOP}^{\mathsf{B}}[u] = \alpha^{\mathsf{B}}(\mathsf{L}^{\mathsf{op}}[u]) \sqcup \alpha^{\mathsf{PI}}(\mathsf{PI}^{\mathsf{op}}[u])$.

The proof is similar to that of Theorem 2. It is deferred to the appendix of [7] due to lack of space.

In the forward case, the set of directly reaching paths could be easily described by a constraint system on sets of paths. The analogous set of directly leaving paths, however, does not appear to have such a simple characterization, because the concurrency effects caused by threads created on these paths have to be tackled. This is hard in combination with procedures, as threads created inside an instance of a procedure can survive termination of that instance. In order to treat this effect, we have experimented with a complex constraint system on sets of pairs of paths. It turned out that this complexity disappears in the abstract version of this constraint system. In order to give a more transparent justification for the resulting abstract constraint systems, we develop – again arguing on the path level – an alternative characterization of $\alpha^{\mathsf{B}}(\mathsf{L}^{\mathsf{op}}[u])$ through a subset of representative paths that is easy to characterize. Thus, again we transfer part of the abstraction to the path level.

More specifically, we only consider directly leaving paths that execute transitions of a created thread immediately after the corresponding spawn transition. From the point of view of the initial thread from which the path is leaving, the transitions of newly created threads are executed as early as possible. Formally, we define the relation $\overset{\cdot}{\Longrightarrow}_{\mathsf{x}} \subseteq \mathsf{Conf} \times \mathcal{L}^* \times \mathsf{Conf}$ by the following rules:

$$
\begin{array}{ll}
c \overset{e}{\Longrightarrow}_{\mathsf{x}} c' & \text{if } c \overset{e}{\longrightarrow}_{x} c' \text{ and } e \text{ is no spawn edge} \\
c \overset{[e]w}{\Longrightarrow}_{\mathsf{x}} c' & \text{if } c \overset{e}{\longrightarrow}_{x} c' \uplus \{[\mathsf{e}_p]\},\ e = \mathsf{spawn}\ p \text{ and } \{[\mathsf{e}_p]\} \overset{w}{\longrightarrow} \_ \\
c \overset{w_1 w_2}{\Longrightarrow}_{\mathsf{x}} c' & \text{if } \exists \hat{c} : c \overset{w_1}{\Longrightarrow}_{\mathsf{x}} \hat{c} \wedge \hat{c} \overset{w_2}{\Longrightarrow}_{\mathsf{x}} c'
\end{array}
$$

Here x selects some set of transition rules, i.e. $x = \mathsf{mp}$ means that $\overset{\cdot}{\longrightarrow}_{\mathsf{mp}}$ is used for $\overset{\cdot}{\longrightarrow}_{x}$. If x is empty, the standard transition relation $\overset{\cdot}{\longrightarrow}$ is used.

The set of representative directly leaving paths is defined by

$$\mathsf{L}^{\mathsf{op}}_{\subseteq}[u] := \{w \mid \exists r, c : \{[\mathsf{e}_{\mathsf{main}}]\} \overset{*}{\longrightarrow} \{[u]r\} \uplus c \wedge \{[u]r\} \overset{w}{\Longrightarrow} \_\} \,.$$

Exploiting structural properties of kill/gen-functions, we can show:

Lemma 5. *For each $u \in N$ we have $\alpha^{\mathsf{B}}(\mathsf{L}^{\mathsf{op}}[u]) = \alpha^{\mathsf{B}}(\mathsf{L}^{\mathsf{op}}_{\subseteq}[u])$.*

Proof. The $\sqsupseteq$-direction is trivial, because we obviously have $\mathsf{L}^{\mathsf{op}}[u] \supseteq \mathsf{L}^{\mathsf{op}}_{\subseteq}[u]$ and α^{B} is monotonic. For the $\sqsubseteq$-direction we consider a directly leaving path $\{[u]r\} \overset{w}{\longrightarrow} \_$ with $w = e_1 \ldots e_n$. Due to the distributivity of L, its abstraction can be written as $f_{w^R}(\bot_L) = \bigsqcup_{1 \leq i \leq n} (\mathsf{gen}_{e_i} \sqcap A_i)$ with $A_i := \mathsf{kill}_{e_1} \sqcap \ldots \sqcap \mathsf{kill}_{e_{i-1}}$.

We show for each edge e_k that the value $\mathsf{gen}_{e_k} \sqcap A_k$ is below $\alpha^{\mathsf{B}}(\mathsf{L}^{\mathsf{op}}_{\subseteq}[u])$. For this, we distinguish whether transition e_k was executed in the initial thread (from stack $[u]r$) or in some spawned thread. To cover the case of a transition e_k of the initial thread, we consider the subpath $w' \in \mathsf{L}^{\mathsf{op}}_{\subseteq}[u]$ of w that makes no transitions of spawned threads at all. We can obtain w' by discarding some transitions from w. Moreover, w' also contains the transition e_k. If we write $f_{w'^R}(\bot_L)$ in a similar form as above, it contains a term $\mathsf{gen}_{e_k} \sqcap A'$, and because we discarded some transitions, we have $A' \sqsupseteq A_k$, and hence $f_{w'^R}(\bot_L) \sqsupseteq \mathsf{gen}_{e_k} \sqcap A' \sqsupseteq \mathsf{gen}_{e_k} \sqcap A_k$.

To cover the case of a transition e_j of a spawned thread, we consider the subpath $w'' \in \mathsf{L}^{\mathsf{op}}_{\subseteq}[u]$ of w that, besides directly leaving ones, only contains transitions of the considered thread. Because e_j occurs as early as possible in w'', the prefix of w'' up to e_j can be derived from the prefix of w up to e_j by discarding some transitions, and again we get $f_{w''^R}(\bot_L) \sqsupseteq \mathsf{gen}_{e_j} \sqcap A_j$. □

We can characterize $\mathsf{L}_{\subseteq}^{\mathsf{op}}$ by the following constraint system:

$$
\begin{array}{lll}
\text{[LS.init]} & \mathsf{LS}[u] \supseteq \{\varepsilon\} & \\
\text{[LS.init2]} & \mathsf{LS}[r_p] \supseteq \{[\mathsf{ret}]\} & \text{for } p \in P \\
\text{[LS.base]} & \mathsf{LS}[u] \supseteq e; \mathsf{LS}[v] & \text{for } e = (u, \mathsf{base}\ \_, v) \in E \\
\text{[LS.call1]} & \mathsf{LS}[u] \supseteq e; \mathsf{LS}[\mathsf{e}_p] & \text{for } (u, \mathsf{call}\ p, v) \in E \\
\text{[LS.call2]} & \mathsf{LS}[u] \supseteq e; \mathsf{S_B}[\mathsf{e}_p]; \mathsf{ret}; \mathsf{LS}[v] & \text{for } (u, \mathsf{call}\ p, v) \in E \\
\text{[LS.spawn]} & \mathsf{LS}[u] \supseteq e; \mathsf{LS}[\mathsf{e}_p]; \mathsf{LS}[v] & \text{for } (u, \mathsf{spawn}\ p, v) \in E \\
 & & \\
\text{[SB.init]} & \mathsf{S_B}[\mathsf{r}_p] \supseteq \{\varepsilon\} & \\
\text{[SB.base]} & \mathsf{S_B}[u] \supseteq e; \mathsf{S_B}[v] & \text{for } e = (u, \mathsf{base}\ \_, v) \in E \\
\text{[SB.call]} & \mathsf{S_B}[u] \supseteq e; \mathsf{S_B}[\mathsf{e}_p]; \mathsf{ret}; \mathsf{S_B}[v] & \text{for } (u, \mathsf{call}\ p, v) \in E \\
\text{[SB.spawn]} & \mathsf{S_B}[u] \supseteq e; \mathsf{LS}[\mathsf{e}_p]; \mathsf{S_B}[v] & \text{for } (u, \mathsf{spawn}\ p, v) \in E \\
 & & \\
\text{[L.leave1]} & \mathsf{L}_{\subseteq}[u] \supseteq \mathsf{S_B}[u]; \mathsf{L}_{\subseteq}[\mathsf{r}_p] & \text{for } u \in N_p \text{ if } u \text{ reachable} \\
\text{[L.leave2]} & \mathsf{L}_{\subseteq}[u] \supseteq \mathsf{LS}[u] & \text{if } u \text{ reachable} \\
\text{[L.ret]} & \mathsf{L}_{\subseteq}[\mathsf{r}_p] \supseteq \mathsf{ret}; \mathsf{L}_{\subseteq}[v] & \text{for } (\_, \mathsf{call}\ p, v) \in E \\
 & & \text{and } p \text{ can terminate}
\end{array}
$$

The LS part of the constraint system characterizes paths from a single control node: $\mathsf{LS}^{\mathsf{op}}[u] := \{w \mid \{[u]\} \overset{w}{\Longrightarrow} \_\}$. The $\mathsf{S_B}$-part characterizes same-level paths from a control node to the return node of the corresponding procedure: $\mathsf{S}_{\mathsf{B}}^{\mathsf{op}}[u] := \{w \mid \exists c' : \{[u^\bullet]\} \overset{w}{\Longrightarrow}_{\mathsf{m}} \{[\mathsf{r}_p^\bullet]\} \uplus c'\}$. It is straightforward to prove $\mathsf{lfp}(\mathsf{LS}) = \mathsf{LS}^{\mathsf{op}}$, $\mathsf{lfp}(\mathsf{S_B}) = \mathsf{S}_{\mathsf{B}}^{\mathsf{op}}$ and $\mathsf{lfp}(\mathsf{L}_{\subseteq}) = \mathsf{L}_{\subseteq}^{\mathsf{op}}$. Using abstract interpretation one gets constraint systems $\mathsf{L}_{\subseteq}{}^{\#}$ over $(L, \subseteq)$ and $\mathsf{LS}^{\#}, \mathsf{S_B}{}^{\#}$ over $(L \overset{\mathsf{mon}}{\to} L, \sqsubseteq)$ with $\mathsf{lfp}(\mathsf{L}_{\subseteq}{}^{\#}) = \alpha^{\mathsf{B}}(\mathsf{lfp}(\mathsf{L}_{\subseteq}))$.

6 Possible Interference

In order to be able to compute the forward and backward MOP-solution, it remains to describe a constraint system based characterization of the possible interference suitable for abstract interpretation. We use the following constraints:

$$
\begin{array}{lll}
\text{[SP.edge]} & \mathsf{SP}[v] \supseteq \mathsf{SP}[u] & \text{for } (u, \mathsf{base}\ \_, v) \in E \text{ or } (u, \mathsf{spawn}\ \_, v) \in E \\
\text{[SP.call]} & \mathsf{SP}[v] \supseteq \mathsf{SP}[u] \cup \mathsf{SP}[\mathsf{r}_q] & \text{for } (u, \mathsf{call}\ q, v) \in E \text{ if } q \text{ can terminate} \\
\text{[SP.spawnt]} & \mathsf{SP}[v] \supseteq \alpha^{\mathsf{E}}(\mathsf{LS}^{\mathsf{op}}[\mathsf{e}_q]) & \text{for } (u, \mathsf{spawn}\ q, v) \in E \\
 & & \\
\text{[PI.reach]} & \mathsf{PI}[u] \supseteq \mathsf{PI}[\mathsf{e}_p] \cup \mathsf{SP}[u] & \text{for } u \in N_p \text{ and } u \text{ reachable} \\
\text{[PI.trans1]} & \mathsf{PI}[\mathsf{e}_q] \supseteq \mathsf{PI}[u] & \text{for } (u, \mathsf{call}\ q, \_) \in E \\
\text{[PI.trans2]} & \mathsf{PI}[\mathsf{e}_q] \supseteq \mathsf{PI}[u] & \text{for } (u, \mathsf{spawn}\ q, \_) \in E \\
\text{[PI.trans3]} & \mathsf{PI}[\mathsf{e}_q] \supseteq \alpha^{\mathsf{E}}(\mathsf{L}_{\subseteq}^{\mathsf{op}}[v]) & \text{for } (u, \mathsf{spawn}\ q, v) \in E
\end{array}
$$

Here, $\alpha^{\mathsf{E}} : \mathcal{P}(\mathcal{L}^*) \to \mathcal{P}(\mathcal{L})$ with $\alpha^{\mathsf{E}}(W) = \{e \mid \exists w, e, w' : w[e]w' \in W\}$ abstracts sets of paths to the sets of transitions contained in the paths. The constraint system PI follows the same-level pattern: SP characterizes the interfering transitions that are enabled by same-level paths. It is straightforward to show $\mathsf{lfp}(\mathsf{SP}) = \mathsf{SP}^{\mathsf{op}}$, with $\mathsf{SP}^{\mathsf{op}}[u] := \{e \mid \exists c, w : \{[\mathsf{e}_p^\bullet]\} \xrightarrow{*}_{\mathsf{m}} \{[u^\bullet]\} \uplus c \xrightarrow{w[e]}_{\mathsf{i\_}}\}$. The constraint [PI.reach] captures that the possible interference at a reachable node u is greater than the possible interference at the beginning of u's procedure and the interference created by same-level paths to u. The [PI.trans1]- and [PI.trans2]-constraints describe that the interference at the entry point of a called or spawned procedure is greater than the interference at the start node of the call resp. spawn edge. The [PI.trans3]-constraint accounts for the interference generated in the spawned thread by the creator thread continuing its execution. Because the creator thread may be inside a procedure, we have to account not only for edges inside the current procedure, but also for edges of procedures the creator thread may return to. These edges are captured by $\alpha^{\mathsf{E}}(\mathsf{L}^{\mathsf{op}}[v]) = \alpha^{\mathsf{E}}(\mathsf{L}_{\subseteq}^{\mathsf{op}}[v])$.

With the ideas described above, it is straightforward to show $\mathsf{lfp}(\mathsf{PI}) = \mathsf{PI}^{\mathsf{op}}$. Abstraction of the PI- and SP-systems is especially simple in this case, as the constraint systems only contain variables and constants. For the abstract versions $\mathsf{SP}^{\#}$ and $\mathsf{PI}^{\#}$, we have $\mathsf{lfp}(\mathsf{SP}^{\#}) = \alpha^{\mathsf{PI}}(\mathsf{lfp}(\mathsf{SP}))$ and $\mathsf{lfp}(\mathsf{PI}^{\#}) = \alpha^{\mathsf{PI}}(\mathsf{lfp}(\mathsf{PI}))$.

Now, we have all pieces to compute the forward and backward MOP-solutions: Combining Theorems 2, 4 and Lemma 5 with the statements about the abstract constraint systems we get

$$\mathsf{MOP}^{\mathsf{F}}[u] = \mathsf{lfp}(\mathsf{R}^{\#})[u] \sqcup \mathsf{lfp}(\mathsf{PI}^{\#})[u] \text{ and } \mathsf{MOP}^{\mathsf{B}}[u] = \mathsf{lfp}(\mathsf{L}_{\subseteq}{}^{\#})[u] \sqcup \mathsf{lfp}(\mathsf{PI}^{\#})[u]\,.$$

The right hand sides are efficiently computable, e.g. by a worklist algorithm [11].

7 Parallel Calls

In this section we discuss the extension to parallel procedure calls. Two procedures that are called in parallel are executed concurrently, but the call does not return until both procedures have terminated.

Flowgraphs. In the flowgraph definition, we replace the $\mathsf{call}\ p$ annotation by the $\mathsf{pcall}\ p_1 \parallel p_2$ annotation, where $p_1, p_2 \in P$ are the procedures called in parallel. Note that there is no loss of expressiveness by assuming that all procedure calls are parallel calls, because instead of calling a procedure p alone, one can call it in parallel with a procedure q_0, where q_0 has only a single node $\mathsf{e}_{q_0} = \mathsf{r}_{q_0}$ with no outgoing edges. To describe a configuration, the notion of a stack is extended from a linear list to a binary tree. While in the case without parallel calls, the topmost node of the stack can make transitions and the other nodes are the stored return addresses, now the leafs of the tree can make transitions and the inner nodes are the stored return addresses. We write u for the tree consisting just of node u, and $u(t, t')$ for the tree with root u with the two successor trees t and t'. The notation $r[t]$ denotes a tree consisting of a subtree t in some context r. The position of t in r is assumed to be fixed, such that writing $r[t]$ and $r[t']$ in the same expression means that t and t' are at the same position in r.

The rule templates for $\overset{\cdot}{\longrightarrow}_{\mathsf{m}}$, $\overset{\cdot}{\longrightarrow}_{\mathsf{i}}$ and $\overset{\cdot}{\longrightarrow}_{\mathsf{p}}$ are refined as follows:

$$\begin{array}{lll}
\text{[base]} & (\{r[u^m]\} \uplus c) \xrightarrow{e}_x (\{r[v^m]\} \uplus c) & e = (u, \mathsf{base}\ a, v) \in E \\
\text{[pcall]} & (\{r[u^m]\} \uplus c) \xrightarrow{e}_x (\{r[v^m(\mathsf{e}_p^\circ, \mathsf{e}_q^\circ)]\} \uplus c) & e = (u, \mathsf{pcall}\ p \parallel q, v) \in E \\
\text{[ret]} & (\{r[v^m(\mathsf{r}_p^\circ, \mathsf{r}_q^\circ)]\} \uplus c) \xrightarrow{\mathsf{ret}}_x (\{r[v^m]\} \uplus c) & p, q \in P \\
\text{[spawn]} & (\{r[u^m]\} \uplus c) \xrightarrow{e}_x (\{r[v^m]\} \uplus \{\mathsf{e}_q^\circ\} \uplus c) & e = (u, \mathsf{spawn}\ q, v) \in E
\end{array}$$

$$\begin{array}{lll}
\text{[c.pushl]} & (\{r[u^\bullet]\} \uplus c) \xrightarrow{e}_{\mathsf{p}} (\{r[v^\circ(\mathsf{e}_p^\bullet, \mathsf{e}_q^\circ)]\} \uplus c) & e = (u, \mathsf{pcall}\ p \parallel q, v) \in E \\
\text{[c.pushr]} & (\{r[u^\bullet]\} \uplus c) \xrightarrow{e}_{\mathsf{p}} (\{r[v^\circ(\mathsf{e}_p^\circ, \mathsf{e}_q^\bullet)]\} \uplus c) & e = (u, \mathsf{pcall}\ p \parallel q, v) \in E \\
\text{[sp.push]} & (\{r[u^\bullet]\} \uplus c) \xrightarrow{e}_{\mathsf{p}} (\{r[v^\circ]\} \uplus \{\mathsf{e}_q^\bullet\} \uplus c) & e = (u, \mathsf{spawn}\ q, v) \in E
\end{array}$$

For the $\overset{\cdot}{\longrightarrow}_{\mathsf{m}}$-relation, we require the position of the processed node u^m resp. subtree $v^m(\mathsf{r}_p^\circ, \mathsf{r}_q^\circ)$ in r to be below a marked node. For the $\overset{\cdot}{\longrightarrow}_{\mathsf{i}}$-relation the position must not be below a marked node. The reference semantics $\overset{\cdot}{\longrightarrow}$ on unmarked configurations is defined by analogous rules.

Forward Analysis. Again we can use the relation $\overset{\cdot}{\longrightarrow}_{\mathsf{mp}}$ to push an initial marker to any reachable node. Although there is some form of synchronization at procedure return now, the $\overset{\cdot}{\longrightarrow}_{\mathsf{m}}$ and $\overset{\cdot}{\longrightarrow}_{\mathsf{i}}$ -relations are defined in such a way that the interfering transitions can again be moved to the end of a reaching path and in analogy to Lemma 1 we get:

Lemma 6. *Given a reaching path* $\{\mathsf{e}_{\mathsf{main}}\} \xrightarrow{w} \{r[u]\} \uplus c$*, there exists paths* w_1, w_2 *with* $w \in w_1 \otimes w_2$ *such that* $\exists \hat{c}, \hat{r} : \{\mathsf{e}_{\mathsf{main}}^\bullet\} \xrightarrow{w_1}_{\mathsf{mp}} \{\hat{r}[u^\bullet]\} \uplus \hat{c} \xrightarrow{w_2}_{\mathsf{i}} \{r[u^\bullet]\} \uplus c$.

Note that the interfering transitions may work not only on the threads in $\hat{c}$, but also on the nodes of $\hat{r}$ that are no predecessors of $u^\bullet$, i.e. on procedures called in parallel that have not yet terminated.

We redefine the directly reaching paths R^{op} and possible interference $\mathsf{PI}^{\mathsf{op}}$ accordingly, and get the same characterization of $\mathsf{MOP}^{\mathsf{F}}$ as in the case without parallel calls (Theorem 2). In S and R, we replace the constraints for call edges with the following constraints for parallel procedure calls:

$$\begin{array}{lll}
\text{[call]} & \mathsf{S}[v] \supseteq \mathsf{S}[u]; e; (\mathsf{S}[\mathsf{r}_p] \otimes \mathsf{S}[\mathsf{r}_q]); \mathsf{ret} & \text{for } e = (u, \mathsf{pcall}\ p \parallel q, v) \in E \\
\text{[callp1]} & \mathsf{R}[\mathsf{e}_p] \supseteq \mathsf{R}[u]; e & \text{for } (u, \mathsf{pcall}\ p \parallel q, \_) \in E \\
\text{[callp2]} & \mathsf{R}[\mathsf{e}_q] \supseteq \mathsf{R}[u]; e & \text{for } (u, \mathsf{pcall}\ p \parallel q, \_) \in E
\end{array}$$

The [call]-constraint accounts for the paths through a parallel call, that are all interleavings of same-level paths through the two procedures called in parallel. For the abstract interpretation of this constraint, we lift the operator $\otimes^\# : (L \overset{\mathsf{mon}}{\rightarrow} L) \times (L \overset{\mathsf{mon}}{\rightarrow} L) \rightarrow (L \overset{\mathsf{mon}}{\rightarrow} L)$ defined by $f \otimes^\# g = f \circ g \sqcup g \circ f$ to sets and use it as precise [13] abstract interleaving operator. The redefined PI constraint system will be described after the backward analysis.

Backward Analysis. For backward analysis, the concept of directly leaving paths has to be generalized: In the case without parallel calls, a directly leaving path is a path from some reachable stack. It is complementary to the possible interference, i.e. the transitions on leaving paths are either interfering or directly leaving transitions. In the case of parallel calls, interference is not only caused by parallel threads, but also by procedures called in parallel. Hence it is not sufficient to just distinguish the thread that reached the node from the other threads, but we have to look inside this thread and distinguish between the procedure reaching the node and the procedures executed in parallel.

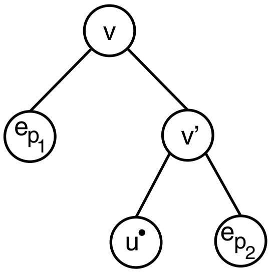


Fig. 2. Sample tree with marked node u

For instance, consider the tree $s = v(\mathsf{e}_{p_1}, v'(u^\bullet, \mathsf{e}_{p_2}))$ that is visualized in Fig. 2 (the $^\circ$-annotations at the unmarked nodes are omitted for better readability). This tree may have been created by a directly reaching path to u. The nodes e_{p_1} and e_{p_2} may execute interfering transitions. The other transitions, that are exactly the directly leaving ones, may either be executed from the node $u^\bullet$, or from the nodes v' and v, if p_1 and p_2 have terminated. To describe the directly leaving transitions separately, we define the function above, that transforms a tree with a marked node $u^\bullet$ by pruning all nodes that are not predecessors of $u^\bullet$ and adding dummy nodes to make the tree binary again. For a subtree that may potentially terminate, i.e. be transformed to a single return node by some transition sequence, a dummy return node u_r is added. If the pruned subtree cannot terminate, a dummy non-return node u_n is added. Both u_r and u_n have no outgoing edges. Assuming, for instance, that procedure p_1 cannot terminate and p_2 can terminate, we would have $\mathsf{above}(s) = v(u_n, v'(u, u_r))$. For technical reasons, above deletes the marker.

With the help of the above-function, we define the directly leaving paths by

$$\mathsf{L}^{\mathsf{op}}[u] := \{w \mid \exists r, c : \{\mathsf{e}^\bullet_{\mathsf{main}}\} \xrightarrow{*}_{\mathsf{mp}} \{r[u^\bullet]\} \uplus c \wedge \{\mathsf{above}(r[u^\bullet])\} \xrightarrow{w} \_\}$$

and show similar to Theorem 4:

Theorem 7. $\mathsf{MOP}^{\mathsf{B}}[u] = \alpha^{\mathsf{B}}(\mathsf{L}^{\mathsf{op}}[u]) \sqcup \alpha^{\mathsf{PI}}(\mathsf{PI}^{\mathsf{op}}[u])$.

The proof formalizes the ideas discussed above. Due to lack of space it is deferred to the appendix of [7].

As in the case without parallel calls, we can characterize the directly leaving paths by a complex constraint system whose abstract version is less complex. So we again perform part of the abstraction on the path level. As in the case without parallel calls, we define the $\overset{\cdot}{\Longrightarrow}$ transition relation, to describe those paths that execute transitions of spawned threads only immediately after the corresponding spawn transition. Transitions executed in parallel due to parallel calls, however, may be executed in any order. Formally, the definition of $\overset{\cdot}{\Longrightarrow}$ looks the same as without parallel calls (we just use trees instead of lists). The definition of $\mathsf{L}^{\mathsf{op}}_{\subseteq}$ changes to: $\mathsf{L}^{\mathsf{op}}_{\subseteq}[u] := \{w \mid \exists r, c : \{\mathsf{e}^\bullet_{\mathsf{main}}\} \xrightarrow{*}_{\mathsf{mp}} \{r[u^\bullet]\} \uplus c \wedge \{\mathsf{above}(r[u^\bullet])\} \overset{w}{\Longrightarrow} \_\}$. The proof of $\alpha^{\mathsf{B}}(\mathsf{L}^{\mathsf{op}}[u]) = \alpha^{\mathsf{B}}(\mathsf{L}^{\mathsf{op}}_{\subseteq}[u])$ (Lemma 5) does not change significantly.

However, we do not know any simple constraint system that characterizes $\mathsf{L}^{\mathsf{op}}_{\subseteq}[u]$. The reason is that there must be constraints that relate the leaving paths before a parallel call to the paths into or through the called procedures. We cannot use sequential composition here, because $\mathsf{L}^{\mathsf{op}}_{\subseteq}$ contains interleavings of procedures called in parallel. But we cannot use interleaving either, because transitions of one parallel procedure might get interleaved arbitrarily with transitions of a thread spawned by the other parallel procedure, which is prohibited by $\overset{\cdot}{\Longrightarrow}$. While it is possible to avoid this problem by working with a more complex definition of $\overset{\cdot}{\Longrightarrow}$, there is a simpler way out. We observe that we need not characterize $\mathsf{L}^{\mathsf{op}}_{\subseteq}[u]$ exactly, but only some set $\mathsf{lfp}(\mathsf{L}_{\subseteq})[u]$ *between* $\mathsf{L}^{\mathsf{op}}_{\subseteq}[u]$ and $\mathsf{L}^{\mathsf{op}}[u]$, i.e. $\mathsf{L}^{\mathsf{op}}[u] \supseteq \mathsf{lfp}(\mathsf{L}_{\subseteq})[u] \supseteq \mathsf{L}^{\mathsf{op}}_{\subseteq}[u]$. From these inclusions, it follows by monotonicity of the abstraction function α^{B}, that $\alpha^{\mathsf{B}}(\mathsf{L}^{\mathsf{op}}[u]) \sqsupseteq \alpha^{\mathsf{B}}(\mathsf{lfp}(\mathsf{L}_{\subseteq})[u]) \sqsupseteq \alpha^{\mathsf{B}}(\mathsf{L}^{\mathsf{op}}_{\subseteq}[u]) = \alpha^{\mathsf{B}}(\mathsf{L}^{\mathsf{op}}[u])$, and thus $\alpha^{\mathsf{B}}(\mathsf{lfp}(\mathsf{L}_{\subseteq})[u]) = \alpha^{\mathsf{B}}(\mathsf{L}^{\mathsf{op}}[u])$.

In order to obtain appropriate constraint systems, we replace in the constraint systems $\mathsf{L}_{\subseteq}$, LS and $\mathsf{S_B}$ the constraints related to call edges as follows:

[LS.init2]	dropped	
[LS.call1]	$\mathsf{LS}[u] \supseteq e; (\mathsf{LS}[\mathsf{e}_{p_1}] \otimes \mathsf{LS}[\mathsf{e}_{p_2}])$	for $(u, \mathsf{pcall}\ p_1 \parallel p_2, v) \in E$
[LS.call2]	$\mathsf{LS}[u] \supseteq e; (\mathsf{S_B}[\mathsf{e}_{p_1}] \otimes \mathsf{S_B}[\mathsf{e}_{p_2}]); \mathsf{ret}; \mathsf{LS}[v]$	for $(u, \mathsf{pcall}\ p_1 \parallel p_2, v) \in E$

[SB.call]	$\mathsf{S_B}[u] \supseteq e; (\mathsf{S_B}[\mathsf{e}_{p_1}] \otimes \mathsf{S_B}[\mathsf{e}_{p_2}]); \mathsf{ret}; \mathsf{S_B}[v]$	for $(u, \mathsf{pcall}\ p_1 \parallel p_2, v) \in E$
[L.ret]	$\mathsf{L}_{\subseteq}[\mathsf{r}_{p_i}] \supseteq \mathsf{ret}; \mathsf{L}_{\subseteq}[v]$	for $(\_, \mathsf{pcall}\ p_1 \parallel p_2, v) \in E$ p_1, p_2 can terminate, $i = 1, 2$

We have to drop the constraint [LS.init2], because in our generalization to parallel calls, the procedure at the root of the tree can never return, while in the model without parallel calls, the procedure at the bottom of the stack may return. The constraints [LS.call1], [LS.call2] and [SB.call] account for any interleaving between the paths into resp. through two procedures called in parallel, even when thereby breaking the atomicity of the transitions of some spawned thread. With the ideas above, it is straightforward to prove the required inclusions $\mathsf{L}^{\mathsf{op}}[u] \supseteq \mathsf{lfp}(\mathsf{L}_{\subseteq}[u]) \supseteq \mathsf{L}^{\mathsf{op}}_{\subseteq}[u]$. As these constraint systems do not contain any new operators, abstract versions can be obtained as usual.

Possible Interference. It remains to modify the constraint system for PI. This is done by replacing the constraints for call edges with the following ones:

[SP.call]	$\mathsf{SP}[v] \supseteq \mathsf{SP}[\mathsf{r}_{p_1}] \cup \mathsf{SP}[\mathsf{r}_{p_2}] \cup \mathsf{SP}[u]$	$(u, \mathsf{pcall}\ p_1 \parallel p_2, v) \in E$ if p_1, p_2 can terminate
[PI.trans1]	$\mathsf{PI}[\mathsf{e}_{p_i}] \supseteq \mathsf{PI}[u]$	$(u, \mathsf{pcall}\ p_1 \parallel p_2, v) \in E, i = 1, 2$
[PI.callmi]	$\mathsf{PI}[\mathsf{e}_{p_i}] \supseteq \alpha^{\mathsf{E}}(\mathsf{LS}^{\mathsf{op}}[\mathsf{e}_{p_{3-i}}])$	$(u, \mathsf{pcall}\ p_1 \parallel p_2, v) \in E, i = 1, 2$

The [SP.call]-constraint now accounts for the interference generated by both procedures called in parallel and [PI.trans1] forwards the interference to both procedures. The [PI.callmi]-constraint has no correspondent in the original PI-system. It accounts for the fact that a procedure p_{3-i} generates interference for p_i in a parallel call $\mathsf{pcall}\ p_1 \parallel p_2$. Again, the necessary soundness and precision proofs as well as abstraction are straightforward.

8 Conclusion

From the results in this paper, we can construct an algorithm for precise kill/gen-analysis of interprocedural flowgraphs with thread creation and parallel procedure calls:

1. Generate the abstract versions of the constraint systems R,PI, $\mathsf{L}_{\subseteq}$ and all dependent constraint systems from the flowgraph.
2. Compute their least solutions.
3. Return the approximation of the $\mathsf{MOP}^{\mathsf{F}}$- and $\mathsf{MOP}^{\mathsf{B}}$-solution respectively, as indicated by Theorem 2, Theorem 4 and Lemma 5.

Let us briefly estimate the complexity of this algorithm: We generate $O(|E|+|P|)$ constraints over $O(|N|)$ variables. If the height of $(L, \sqsubseteq)$ is bounded by $h(L)$ and a lattice operation (join, compare, assign) needs time $O(op)$, the algorithm needs time $O((|E| * h(L) + |N|) * op)$ if a worklist algorithm [11] is used in Step 2. A prototype implementation of our algorithm for forward problems has been constructed in [6]. The algorithm may be extended to treat local variables using well-known techniques; see e.g. [13].

Compared to related work, our contributions are the following: Generalizing [13], we treat thread creation in addition to parallel procedure calls and handle backward analysis completely. Compared to [1], our analysis computes information for all program points in linear time, while the automata based algorithm of [1] needs at least linear time *per* program point. Moreover, representing powersets by bitvectors as usual, we can exploit efficient bitvector operations, while the algorithm of [1] needs to be iterated for each bit.

Like other related work [5,13,3,4,9,1], we do not handle synchronization such as message passing. In presence of such synchronization, we still get a correct (but weak) approximation. There are limiting undecidability results [12], but further research has to be done to increase approximation quality in presence of synchronization. Also extensions to more complex domains, e.g. analysis of transitive dependences as studied in [9] for parallel calls, have to be investigated.

Acknowledgment. We thank Helmut Seidl and Bernhard Steffen for interesting discussions on analysis of parallel programs and the anonymous reviewers for their helpful comments.

References

1. 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, Springer, Heidelberg (2005)
2. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. of POPL'77, Los Angeles, California, pp. 238–252. ACM Press, New York (1977)
3. Esparza, J., Knoop, J.: An automata-theoretic approach to interprocedural dataflow analysis. In: Thomas, W. (ed.) ETAPS 1999 and FOSSACS 1999. LNCS, vol. 1578, pp. 14–30. Springer, Heidelberg (1999)

4. Esparza, J., Podelski, A.: Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In: Proc. of POPL'00, pp. 1–11. Springer, Heidelberg (2000)
5. Knoop, J., Steffen, B., Vollmer, J.: Parallelism for free: Efficient and optimal bitvector analyses for parallel programs. TOPLAS 18(3), 268–299 (1996)
6. Lammich, P.: Fixpunkt-basierte optimale Analyse von Programmen mit Thread-Erzeugung. Master's thesis, University of Dortmund (May 2006)
7. Lammich, P., Müller-Olm, M.: Precise fixpoint-based analysis of programs with thread-creation. Version with appendix. Available from `http://cs.uni-muenster.de/u/mmo/pubs/`
8. Lammich, P., Müller-Olm, M.: Precise fixed point based analysis of programs with thread-creation. In: Proc. of MEMICS 2006, pp. 91–98. Faculty of Information Technology, Brno University of Technology (2006)
9. Müller-Olm, M.: Precise interprocedural dependence analysis of parallel programs. Theor. Comput. Sci. 311(1-3), 325–388 (2004)
10. Müller-Olm, M., Seidl, H.: On optimal slicing of parallel programs. In: Proc. of STOC'01, pp. 647–656. ACM Press, New York, NY, USA (2001)
11. Nielson, F., Nielson, H., Hankin, C.: Principles of Program Analysis. Springer, Heidelberg (1999)
12. Ramalingam, G.: Context-sensitive synchronization-sensitive analysis is undecidable. TOPLAS 22(2), 416–430 (2000)
13. Seidl, H., Steffen, B.: Constraint-Based Inter-Procedural Analysis of Parallel Programs. Nordic Journal of Computing (NJC) 7(4), 375–400 (2000)

Automatic Derivation of Compositional Rules in Automated Compositional Reasoning[*]

Bow-Yaw Wang

Institute of Information Science
Academia Sinica, Taiwan
bywang@iis.sinica.edu.tw

Abstract. Soundness of compositional reasoning rules depends on computational models and sometimes is rather involved. Verifiers are therefore forced to mould their problems into a handful of sound compositional rules known to them. In this paper, a syntactic approach to establishing soundness of compositional rules in automated compositional reasoning is presented. Not only can our work justify all compositional rules known to us, but also derive new circular rules by intuitionistic reasoning automatically. Invertibility issues are also briefly discussed in the paper.

1 Introduction

One of the most effective techniques to alleviate the state-explosion problem in formal verification is compositional reasoning. The technique divides compositions and conquers the verification problem by parts. The decomposition however cannot be done naively. Oftentimes, components function correctly only in specific contexts; they may not work separately. Assume-guarantee reasoning circumvents the problem by introducing environmental assumptions. Nevertheless, making proper environmental assumptions requires clairvoyance. It is so tedious a task that one would like to do without.

In [4], the problem is solved by a novel application of the L^* learning algorithm. Consider, for example, the following assume-guarantee rule where $M \models P$ denotes that the system M satisfies the property P.

$$\frac{M_0 \| A \models P \qquad M_1 \models A}{M_0 \| M_1 \models P}$$

To apply the rule, the new paradigm constructs an assumption A satisfying all premises via automated supervised learning. Verifiers need not construe environmental assumptions in compositional rules anymore.

Nevertheless few compositional rules have been established in automated compositional reasoning. Since proofs of their soundness are essentially tedious case analysis, verifiers may be reluctant to develop new rules lest introducing flaws

[*] The work is partly supported by NSC grands 95-3114-P-001-002-Y02, 95-2221-E-001-024-MY3, and the SISARL thematic project of Academia Sinica.

in the paradigm. Subsequently, all verification tasks must be moulded into a handful of compositional rules available in automated compositional reasoning. The effectiveness and applicability of the new paradigm are therefore impeded.

In this paper, a proof-theoretic approach for establishing soundness of rules in automated compositional reasoning is developed. We simply observe that regular languages form a Boolean algebra. The proof system LK for classical logic can hence be used to deduce relations among regular sets syntactically.

But classical logic has its limitation. Consider the following rule [2].

$$\frac{M_0\|P_1 \models P_0 \qquad P_0\|M_1 \models P_1}{M_0\|M_1 \models P_0\|P_1}$$

If we treat compositions and satisfactions as conjunctions and implications respectively, it is easy to see that the rule is not sound in the Boolean domain. Hence sound proof systems for Boolean algebra cannot derive the circular rule.

Following Abadi and Plotkin's work in [1], we show that non-empty, prefix-closed regular languages form a Heyting algebra. Hence the proof system LJ for intuitionistic logic can be used to deduce relations among them. Moreover, a circular inference rule in [1] is shown to be sound in our settings. After adding it to the system LJ, we are able to derive the soundness of the aforementioned circular compositional rule syntactically.

With the help of modern proof assistants, we can in fact justify compositional rules automatically. For the classical interpretation, the proof assistant Isabelle [10] is used to establish the soundness of all compositional rules in [4,3]. The proof assistant Coq [11] proves the soundness of the circular compositional rule [2] and variants of assume-guarantee rules in [4,3] by intuitionistic reasoning. The proof search engines in both tools are able to justify all rules without human intervention. Verifiers are hence liberated from tedious case analysis in proofs of soundness and can establish their own rules effortlessly.

Many compositional reasoning rules have been proposed in literature (for a comprehensive introduction, see [5]). The present work focuses on the rules used in automated compositional reasoning via learning [4,3]. Instead of proposing new rules for the paradigm, a systematic way to establishing compositional rules is developed. Since it is impossible to enumerate all rules for various scenarios, we feel our work could be more useful to practitioners.

Although we are motivated by automated compositional reasoning, our techniques borrow extensively from Abadi and Plotkin [1]. More recently, semantic analysis of circular compositional reasoning rules and intuitionistic linear temporal logic have also been investigated in [8,9]. There is, nonetheless, a fundamental difference from previous works. An automata-theoretic technique is used in the present work. In addition to algebraic semantics, we give a construction of the automata for intuitionistic implication. Our construction therefore shows that the algebraic semantics is not only computable but also closed under all logical operations. In contrast, computability and closure property are not discussed previously. They are in fact crucial in automated compositional reasoning because the L^* algorithm can only generate descriptions for regular languages.

The paper is organized as follows. After the preliminaries in Section 2, a classical interpretation of propositional logic over regular languages and its limitation are presented in Section 3. The intuitionistic interpretation is then followed in Section 4. Applications are illustrated in Section 5. We briefly discuss the invertibility issues in Section 6. Finally, we conclude the paper in Section 7.

2 Preliminaries

We begin the definitions of algebraic models and their properties. They are followed by the descriptions of proof systems. Elementary results in finite automata theory are also recalled. For more detailed exposition, please refer to [7,13,6].

A *partially ordered set* $\mathcal{P} = (P, \leq)$ consists of a set P and a reflexive, anti-symmetric, and transitive binary relation $\leq$ over P. Given a set $A \subseteq P$, an element u is an *upper bound* of A if $a \leq u$ for all $a \in A$. The element u is a *least upper bound* if u is an upper bound of A and $u \leq v$ for any upper bound v of A. Lower bounds and greatest lower bounds of A can be defined symmetrically. Since $\leq$ is anti-symmetric, it is straightforward to verify that least upper bounds and greatest lower bounds for a fixed set are unique.

Definition 1. *A* lattice $\mathcal{L} = (L, \leq, \sqcup, \sqcap)$ *is a partially ordered set where the least upper bound* $(a \sqcup b)$ *and the greatest lower bound* $(a \sqcap b)$ *exist for any* $\{a, b\}$ *with* $a, b \in L$.

A lattice $\mathcal{L} = (L, \leq, \sqcup, \sqcap)$ is *distributive* if $a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$ and $a \sqcup (b \sqcap c) = (a \sqcup b) \sqcap (a \sqcup c)$ for $a, b, c \in L$. $\mathcal{L}$ is *bounded* if it has a *unit* $1 \in L$ and a *zero* $0 \in L$ such that $0 \leq a$ and $a \leq 1$ for all $a \in L$.

In a bounded lattice, b is a *complement* of a if $a \sqcup b = 1$ and $a \sqcap b = 0$. A bounded lattice is *complemented* if each element has a complement. It can be shown that complements are unique in any bounded distributive lattice. A Boolean algebra is but a complemented distributive lattice.

Definition 2. *A* Boolean algebra $\mathcal{B} = (B, \leq, \sqcup, \sqcap, -, 0, 1)$ *is a complemented distributive lattice where*

- $a \sqcup b$ *and* $a \sqcap b$ *are the least upper bound and the greatest lower bound of* a *and* b *respectively;*
- $-a$ *is the complement of* a*; and*
- 0 *and* 1 *are its zero and unit respectively.*

The complement of a can be viewed as the greatest element incompatible with a (that is, the greatest c such that $a \sqcap c = 0$). The idea can be generalized to define complements relative to arbitrary elements as follows.

Definition 3. *Let* $\mathcal{L} = (L, \leq, \sqcup, \sqcap)$ *be a lattice. For any* a *and* b *in* L*, a* pseudo-complement *of* a relative *to* b *is an element* p *in* L *such that*

$$\text{for all } c, c \leq p \text{ if and only if } a \sqcap c \leq b.$$

Since a lattice is also a partially ordered set, pseudo-complements of a relative to b are in fact unique. We hence write $a \Rightarrow b$ for the pseudo-complement of a relative to b. A lattice is *relatively pseudo-complemented* if the pseudo-complement of a relative to b exists for all a and b. It can be shown that the unit exists in any relatively pseudo-complemented lattice. A Heyting algebra can now be defined formally as a relatively pseudo-complemented lattice with a zero.

Definition 4. *A* Heyting algebra $\mathcal{H} = (H, \leq, \sqcup, \sqcap, \Rightarrow, 0, 1)$ *is a relatively pseudo-complemented lattice with a zero where*

– *$a \sqcup b$ and $a \sqcap b$ are the least upper bound and the greatest lower bound of a and b respectively;*
– *$a \Rightarrow b$ is the pseudo-complement of a relative to b; and*
– *0 and 1 are its zero and unit respectively.*

The following lemma relates pseudo-complements with the partial order in a lattice. It is very useful when the syntactic deduction and semantic interpretation are connected later in our exposition.

Lemma 1. *Let $\mathcal{L} = (L, \leq, \sqcup, \sqcap)$ be a relatively pseudo-complemented lattice. Then $a \Rightarrow b = 1$ if and only if $a \leq b$.*[1]

We will consider both classical and intuitionistic propositional logics in this work. Given a set PV of *propositional variables* and $P \in PV$, the syntax of *propositional formulae* is defined as follows.

$$\varphi = P \big| \bot \big| \varphi \vee \varphi \big| \varphi \wedge \varphi \big| \varphi \to \varphi$$

We will use φ, ψ to range over propositional formulae and write $\neg\varphi$ and $\varphi \leftrightarrow \varphi'$ for $\varphi \to \bot$ and $(\varphi \to \varphi') \wedge (\varphi' \to \varphi)$ respectively.

Let Γ and Δ be finite sets of propositional formulae. A *sequent* is of the form $\Gamma \vdash_\bullet \Delta$. For simplicity, we write $\varphi, \Gamma \vdash_\bullet \Delta, \varphi'$ for $\{\varphi\} \cup \Gamma \vdash_\bullet \Delta \cup \{\varphi'\}$. An *inference rule* in a proof system is represented by

$$\ell \, \frac{\Gamma_0 \vdash_\bullet \Delta_0 \qquad \cdots \qquad \Gamma_n \vdash_\bullet \Delta_n}{\Gamma \vdash_\bullet \Delta}$$

where ℓ is the label of the rule, $\Gamma_0 \vdash_\bullet \Delta_0, \ldots, \Gamma_n \vdash_\bullet \Delta_n$ its *premises*, and $\Gamma \vdash_\bullet \Delta$ its *conclusion*. A *proof tree* for the sequent $\Gamma \vdash_\bullet \Delta$ is a tree rooted at $\Gamma \vdash_\bullet \Delta$ and constructed according to inference rules in a proof system. Proof systems offer a syntactic way to derive valid formulae. Gentzen gives the proof system LK for classical first-order logic. Its propositional fragment LK_0 is shown in Figure 2.[2] A proof tree in system LK_0 can be found in Figure 1.

Let $\mathcal{B} = (B, \leq, \sqcup, \sqcap, -, 0, 1)$ be a Boolean algebra. Define a *valuation* ρ in $\mathcal{B}$ to be a mapping from PV to B. The valuation $[\![\varphi]\!]_K^\rho$ of a propositional formula φ is defined as follows.

[1] Readers are referred to [14] for detailed proofs and more examples.
[2] Figure 2 is in fact a variant of the system LK_0, see [13].

$$\begin{array}{rlrl}
[\![P]\!]^\rho_K &= \rho(P) \text{ for } P \in PV & [\![\bot]\!]^\rho_K &= 0 \\
[\![\varphi \vee \varphi']\!]^\rho_K &= [\![\varphi]\!]^\rho_K \sqcup [\![\varphi']\!]^\rho_K & [\![\varphi \wedge \varphi']\!]^\rho_K &= [\![\varphi]\!]^\rho_K \sqcap [\![\varphi']\!]^\rho_K \\
[\![\varphi \rightarrow \varphi']\!]^\rho_K &= -[\![\varphi]\!]^\rho_K \sqcup [\![\varphi']\!]^\rho_K & &
\end{array}$$

Given a Boolean algebra $\mathcal{B} = (B, \leq, \sqcup, \sqcap, -, 0, 1)$, a valuation ρ in $\mathcal{B}$, a propositional formula φ, and a set of propositional formulae Γ, we define $\mathcal{B}, \rho \models_K \varphi$ if $[\![\varphi]\!]^\rho_K = 1$ and $\mathcal{B}, \rho \models_K \Gamma$ if $\mathcal{B}, \rho \models_K \varphi$ for all $\varphi \in \Gamma$. Moreover, $\Gamma \models_K \varphi$ if $\mathcal{B}, \rho \models_K \Gamma$ implies $\mathcal{B}, \rho \models_K \varphi$ for all $\mathcal{B}, \rho$. The following theorem states that the system LK_0 is both sound and complete with respect to Boolean algebra.

Theorem 1. *Let Γ be a set of propositional formulae and φ a propositional formula. $\Gamma \vdash_K \varphi$ if and only if $\Gamma \models_K \varphi$.*

In contrast to classical logic, intuitionistic logic does not admit the law of excluded middle ($\varphi \vee \neg\varphi$). Philosophically, intuitionistic logic is closely related to constructivism. Its proof system, however, can be obtained by a simple restriction on the system LK: all sequents have exactly one formula at their right-hand side.[3] Figure 3 shows the propositional fragment of the system LJ. A sample proof tree in system LJ_0 is shown in Figure 1.

Let $\mathcal{H} = (H, \leq, \sqcup, \sqcap, \Rightarrow, 0, 1)$ be a Heyting algebra. A *valuation* η in $\mathcal{H}$ is a mapping from PV to H. Define the valuation $[\![\varphi]\!]^\eta_J$ as follows.

$$\begin{array}{rlrl}
[\![P]\!]^\eta_J &= \eta(P) \text{ for } P \in PV & [\![\bot]\!]^\eta_J &= 0 \\
[\![\varphi \vee \varphi']\!]^\eta_J &= [\![\varphi]\!]^\eta_J \sqcup [\![\varphi']\!]^\eta_J & [\![\varphi \wedge \varphi']\!]^\eta_J &= [\![\varphi]\!]^\eta_J \sqcap [\![\varphi']\!]^\eta_J \\
[\![\varphi \rightarrow \varphi']\!]^\eta_J &= [\![\varphi]\!]^\eta_J \Rightarrow [\![\varphi']\!]^\eta_J & &
\end{array}$$

Let $\mathcal{H} = (H, \leq, \sqcup, \sqcap, \Rightarrow, 0, 1)$ be a Heyting algebra, η a valuation, φ a propositional formula, and Γ a set of propositional formulae. The following satisfaction relations are defined similarly: $\mathcal{H}, \rho \models_J \varphi$ if $[\![\varphi]\!]_\rho = 1$, $\mathcal{H}, \rho \models_J \Gamma$ if $\mathcal{H}, \rho \models_J \varphi$ for all $\varphi \in \Gamma$, and $\Gamma \models_J \varphi$ if $\mathcal{H}, \rho \models_J \Gamma$ implies $\mathcal{H}, \rho \models_J \varphi$ for all $\mathcal{H}, \rho$. The system LJ_0 is both sound and complete with respect to Heyting algebra.

Theorem 2. *Let Γ be a set of propositional formulae and φ a propositional formula. $\Gamma \vdash_J \varphi$ if and only if $\Gamma \models_J \varphi$.*

Fix a set Σ of *alphabets*. A *string* is a finite sequence $a_1 a_2 \cdots a_n$ such that $a_i \in \Sigma$ for $1 \leq i \leq n$. The set of strings over Σ is denoted by Σ^*. Given a string $w = a_1 a_2 \cdots a_n$, its *length* (denoted by $|w|$) is n. The *empty string* ϵ is the string of length 0. Moreover, the *i-prefix* of $w = a_1 a_2 \cdots a_{|w|}$, denoted by $w\downarrow_i$, is the substring $a_1 a_2 \cdots a_i$. We define $w\downarrow_0$ to be ϵ for any $w \in \Sigma^*$. A *language* over Σ is a subset of Σ^*. Let $L \subseteq \Sigma^*$ be a language. Define its *complement* $\overline{L}$ to be $\Sigma^* \setminus L$. L is *prefix-closed* if for any string $w \in L$, $w\downarrow_i \in L$ for all $0 \leq i \leq |w|$.

Definition 5. *A* finite state automaton M *is a tuple* $(Q, q_0, \longrightarrow, F)$ *where*

– Q *is a non-empty finite set of* states*;*

[3] System LJ_0 in Figure 3 is again a variant of Gentzen's system and is not obtained by the restriction.

$$\text{R}{\to}\,\dfrac{\text{L}{\wedge}\,\dfrac{\text{L}{\to}\,\dfrac{\text{R}{\wedge}\,\dfrac{\text{R}{\to}\,\dfrac{\text{RW}\,\dfrac{\text{LW}\,\dfrac{\text{LW}\,\dfrac{\text{L}{\to}\,\dfrac{\text{RW}\,\dfrac{\text{R}{\wedge}\,\dfrac{\text{Ax}\,\dfrac{}{M_0, A_0 \vdash_K M_0} \qquad \text{Ax}\,\dfrac{}{M_0, A_0 \vdash_K A_0}}{M_0, A_0 \vdash_K M_0 \wedge A_0}}{M_0, A_0 \vdash_K P, M_0 \wedge A_0} \qquad \text{Ax}\,\dfrac{}{M_0, A_0, P \vdash_K P}}{M_0 \wedge A_0 \to P, M_0, A_0 \vdash_K P}}{M_0 \wedge A_0 \to P, M_0, M_1, A_0 \vdash_K P}}{M_0 \wedge A_0 \to P, M_1 \wedge A_1 \to P, M_0, M_1, A_0 \vdash_K P}}{M_0 \wedge A_0 \to P, M_1 \wedge A_1 \to P, M_0, M_1, A_0 \vdash_K \bot, P}}{M_0 \wedge A_0 \to P, M_1 \wedge A_1 \to P, M_0, M_1 \vdash_K \neg A_0, P} \qquad \dfrac{\vdots}{\begin{array}{c} M_0 \wedge A_0 \to P, \\ M_1 \wedge A_1 \to P, M_0, M_1 \end{array} \vdash_K \neg A_1, P}}{M_0 \wedge A_0 \to P, M_1 \wedge A_1 \to P, M_0, M_1 \vdash_K \neg A_0 \wedge \neg A_1, P} \qquad \text{Ax}\,\dfrac{}{\begin{array}{c} M_0 \wedge A_0 \to P, \\ M_1 \wedge A_1 \to P, \\ P, M_0, M_1 \end{array} \vdash_K P}}{M_0 \wedge A_0 \to P, M_1 \wedge A_1 \to P, \neg A_0 \wedge \neg A_1 \to P, M_0, M_1 \vdash_K P}}{M_0 \wedge A_0 \to P, M_1 \wedge A_1 \to P, \neg A_0 \wedge \neg A_1 \to P, M_0 \wedge M_1 \vdash_K P}}{M_0 \wedge A_0 \to P, M_1 \wedge A_1 \to P, \neg A_0 \wedge \neg A_1 \to P \vdash_K M_0 \wedge M_1 \to P}$$

(a) A Proof Tree in LK_0

$$\text{R}{\to}\,\dfrac{\text{L}{\wedge}\,\dfrac{\text{L}{\to}\,\dfrac{\text{R}{\wedge}\,\dfrac{\text{R}{\to}\,\dfrac{\text{LW}\,\dfrac{\text{LW}\,\dfrac{\text{L}{\to}\,\dfrac{\text{R}{\wedge}\,\dfrac{\text{Ax}\,\dfrac{}{M_1, P_0 \vdash_J P_0} \qquad \text{Ax}\,\dfrac{}{M_1, P_0 \vdash_J M_1}}{M_1, P_0 \vdash_J P_0 \wedge M_1} \qquad \text{Ax}\,\dfrac{}{P_1, M_1, P_0 \vdash_J P_1}}{P_0 \wedge M_1 \to P_1, M_1, P_0 \vdash_J P_1}}{P_0 \wedge M_1 \to P_1, M_0, M_1, P_0 \vdash_J P_1}}{M_0 \wedge P_1 \to P_0, P_0 \wedge M_1 \to P_1, M_0, M_1, P_0 \vdash_J P_1}}{M_0 \wedge P_1 \to P_0, P_0 \wedge M_1 \to P_1, M_0, M_1 \vdash_J P_0 \to P_1} \qquad \dfrac{\vdots}{\begin{array}{c} M_0 \wedge P_1 \to P_0, \\ P_0 \wedge M_1 \to P_1, \\ M_0, M_1 \end{array} \vdash_J P_1 \to P_0}}{M_0 \wedge P_1 \to P_0, P_0 \wedge M_1 \to P_1, M_0, M_1 \vdash_J (P_0 \to P_1) \wedge (P_1 \to P_0)} \qquad \text{Ax}\,\dfrac{}{\begin{array}{c} P_0 \wedge P_1, \\ M_0 \wedge P_1 \to P_0, \\ P_0 \wedge M_1 \to P_1, \\ M_0, M_1 \end{array} \vdash_J P_0 \wedge P_1}}{(P_0 \to P_1) \wedge (P_1 \to P_0) \to P_0 \wedge P_1, M_0 \wedge P_1 \to P_0, P_0 \wedge M_1 \to P_1, M_0, M_1 \vdash_J P_0 \wedge P_1}}{(P_0 \to P_1) \wedge (P_1 \to P_0) \to P_0 \wedge P_1, M_0 \wedge P_1 \to P_0, P_0 \wedge M_1 \to P_1, M_0 \wedge M_1 \vdash_J P_0 \wedge P_1}}{(P_0 \to P_1) \wedge (P_1 \to P_0) \to P_0 \wedge P_1, M_0 \wedge P_1 \to P_0, P_0 \wedge M_1 \to P_1 \vdash_J M_0 \wedge M_1 \to P_0 \wedge P_1}$$

(b) A Proof Tree in LJ_0

Fig. 1. Proof Trees in LK_0 and LJ_0

$$\text{Ax}\ \frac{}{P, \Gamma \vdash_K \Delta, P}\ P \in PV \qquad \text{L}\bot\ \frac{}{\bot, \Gamma \vdash_K}$$

$$\text{LW}\ \frac{\Gamma \vdash_K \Delta}{\varphi, \Gamma \vdash_K \Delta} \qquad \text{RW}\ \frac{\Gamma \vdash_K \Delta}{\Gamma \vdash_K \Delta, \varphi}$$

$$\text{L}\wedge\ \frac{\varphi, \varphi', \Gamma \vdash_K \Delta}{\varphi \wedge \varphi', \Gamma \vdash_K \Delta} \qquad \text{R}\wedge\ \frac{\Gamma \vdash_K \Delta, \varphi \qquad \Gamma \vdash_K \Delta, \varphi'}{\Gamma \vdash_K \Delta, \varphi \wedge \varphi'}$$

$$\text{L}\vee\ \frac{\varphi, \Gamma \vdash_K \Delta \qquad \varphi', \Gamma \vdash_K \Delta}{\varphi \vee \varphi', \Gamma \vdash_K \Delta} \qquad \text{R}\vee\ \frac{\Gamma \vdash_K \Delta, \varphi, \varphi'}{\Gamma \vdash_K \Delta, \varphi \vee \varphi'}$$

$$\text{L}\!\rightarrow\ \frac{\Gamma \vdash_K \Delta, \varphi \qquad \varphi', \Gamma \vdash_K \Delta}{\varphi \rightarrow \varphi', \Gamma \vdash_K \Delta} \qquad \text{R}\!\rightarrow\ \frac{\varphi, \Gamma \vdash_K \Delta, \varphi'}{\Gamma \vdash_K \Delta, \varphi \rightarrow \varphi'}$$

Fig. 2. The System LK_0

$$\text{Ax}\ \frac{}{P, \Gamma \vdash_J P}\ P \in PV \qquad \text{L}\bot\ \frac{}{\bot, \Gamma \vdash_J \psi}$$

$$\text{LW}\ \frac{\Gamma \vdash_J \psi}{\varphi, \Gamma \vdash_J \psi}$$

$$\text{L}\wedge\ \frac{\varphi, \varphi', \Gamma \vdash_J \psi}{\varphi \wedge \varphi', \Gamma \vdash_J \psi} \qquad \text{R}\wedge\ \frac{\Gamma \vdash_J \psi \qquad \Gamma \vdash_J \psi'}{\Gamma \vdash_J \psi \wedge \psi'}$$

$$\text{L}\vee\ \frac{\varphi, \Gamma \vdash_J \psi \qquad \varphi', \Gamma \vdash_J \psi}{\varphi \vee \varphi', \Gamma \vdash_J \psi} \qquad \text{R}\vee\ \frac{\Gamma \vdash_J \psi_i}{\Gamma \vdash_J \psi_0 \vee \psi_1}\ (i = 0, 1)$$

$$\text{L}\!\rightarrow\ \frac{\Gamma \vdash_J \varphi \qquad \varphi', \Gamma \vdash_J \psi}{\varphi \rightarrow \varphi', \Gamma \vdash_J \psi} \qquad \text{R}\!\rightarrow\ \frac{\varphi, \Gamma \vdash_J \psi}{\Gamma \vdash_J \varphi \rightarrow \psi}$$

Fig. 3. The System LJ_0

– $q_0 \in Q$ *is its* initial state*;*
– $\longrightarrow \subseteq Q \times \Sigma \times Q$ *is the total* transition relation*; and*
– $F \subseteq Q$ *is the* accepting states.

We say a finite state automaton is *deterministic* if $\longrightarrow$ is a total function from $Q \times \Sigma$ to Q. It is known that determinism does not change the expressiveness of finite state automata [7]. For clarity, we write $q \xrightarrow{a} q'$ for $(q, a, q') \in \longrightarrow$. A *run* of a string $w = a_1 a_2 \cdots a_n$ in M is a finite alternating sequence $q_0 a_1 q_1 a_2 \cdots q_{n-1} a_n q_n$ such that $q_i \xrightarrow{a_{i+1}} q_{i+1}$ for $0 \leq i < n$; it is *accepting* if $q_n \in F$. We say a string w is *accepted* by M if there is an accepting run of w in M. The *language accepted by* M, $L(M)$, is the set of strings accepted by M. A language $L \subseteq \Sigma^*$ is *regular* if there is a finite state automaton M such that $L = L(M)$.

Theorem 3. *[7] Let L and L' be regular. Then $L \cup L'$, $L \cap L'$, and $\overline{L}$ are regular.*

3 Classical Interpretation

It is trivial to see that regular languages form a Boolean algebra. More formally, define $R = \{L \subseteq \Sigma^* : L \text{ is regular }\}$. We have the following theorem.

Theorem 4. *Let $\mathcal{R} = (R, \subseteq, \cup, \cap, \overline{\bullet}, \emptyset, \Sigma^*)$. $\mathcal{R}$ is a Boolean algebra.*

To illustrate the significance of Theorem 4, let us consider the following scenario. Suppose we are given five regular languages M_0, M_1, A_0, A_1, and P. Further, assume $M_0 \cap A_0 \subseteq P$, $M_1 \cap A_1 \subseteq P$, and $\overline{A}_0 \cap \overline{A}_1 \subseteq P$. We can deduce $M_0 \cap M_1 \subseteq P$ as follows. First, consider the valuation ρ that assigns propositional variables to regular languages of the same name. Suppose there is a proof tree for the following sequent.

$$M_0 \wedge A_0 \rightarrow P, M_1 \wedge A_1 \rightarrow P, \neg A_0 \wedge \neg A_1 \rightarrow P \vdash_K M_0 \wedge M_1 \rightarrow P.$$

Theorem 1 and 4 ensure that if $\mathcal{R}, \rho \models_K M_0 \wedge A_0 \rightarrow P$, $\mathcal{R}, \rho \models_K M_1 \wedge A_1 \rightarrow P$, and $\mathcal{R}, \rho \models_K \neg A_0 \wedge \neg A_1 \rightarrow P$, then $\mathcal{R}, \rho \models_K M_0 \wedge M_1 \rightarrow P$. Lemma 1 gives exactly what we want in $\mathcal{R}$. Hence the proof tree in Figure 1 (a) suffices to show $M_0 \cap M_1 \subseteq P$. Note that we do not make semantic arguments in the analysis. Instead, Theorem 1 allows us to derive semantic property $M_0 \cap M_1 \subseteq P$ by manipulating sequents syntactically.

3.1 Limitation of Classical Interpretation

Consider the following circular inference rule proposed in [1].

$$\frac{}{\vdash [(E \rightarrow M) \wedge (M \rightarrow E)] \rightarrow M}$$

It is easy to see that the valuation of the conclusion is 0 by taking $E = M = 0$ in any Boolean algebra. Since the system LK_0 is sound for Boolean algebra, we conclude that the rule is not derivable. But it does not imply that the rule is not sound in other semantics. To give insights to the intuitionistic interpretation, it is instructive to see how the rule fails in non-trivial cases.

Consider the automata M and E in Figure 4. Let the valuation ρ be that $\rho(E) = L(E)$ and $\underline{\rho(M)} = L(M)$. Observe that the string $bd \notin L(M)$. Hence $bd \in \rho(M \rightarrow E) = \overline{L(M)} \cup L(E)$. Similarly, $bd \in \rho(E \rightarrow M)$. We have $\rho(M \rightarrow E) \cap \rho(E \rightarrow M) \not\subseteq \rho(M)$. Hence $\nvdash [(E \rightarrow M) \wedge (M \rightarrow E)] \rightarrow M$ by Lemma 1. Note that both $L(M)$ and $L(E)$ are non-empty and prefix-closed.

In classical interpretation, the valuation of $E \rightarrow M$ is defined as $\overline{\rho(E)} \cup \rho(M)$. Hence $\rho(M \rightarrow E) \cap \rho(E \rightarrow M) = (\overline{\rho(M)} \cap \overline{\rho(E)}) \cup (\rho(M) \cap \rho(E))$. The problem arises exactly when $\overline{\rho(E)} \cap \overline{\rho(M)}$ is not empty. One may suspect that the valuation

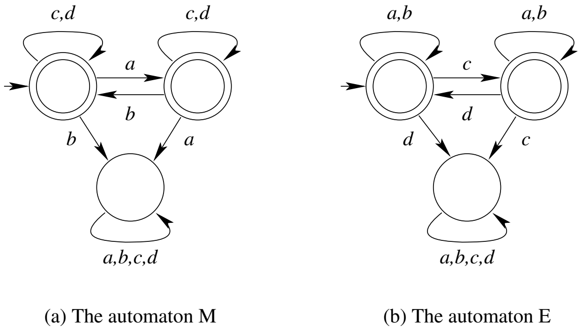


Fig. 4. Limitation of Classical Interpretation

of $E \to M$ is defined too liberally in classical interpretation and resort to a more conservative interpretation. This is indeed the approach taken by Abadi and Plotkin and followed in this work.

4 Interpretation à la Abadi and Plotkin

In order to admit circular compositional rules, an interpretation inspired by [1] is developed here. Mimicking the definition of relative pseudo-complements in [1], we show that non-empty, prefix-closed regular languages form a Heyting algebra. The following lemma gives the zero element and simple closure properties.

Lemma 2. *Let K and L be non-empty, prefix-closed regular languages. Then*

- $\epsilon \in L$ *if and only if* $L \neq \emptyset$*; and*
- $K \cap L$ *and* $K \cup L$ *are non-empty, prefix-closed regular languages.*

For relative pseudo-complements, we follow the definition in [1]. Note that the following definition does not allude to closure properties. In order to define a Heyting algebra, one must show that non-empty, prefix-closed regular languages are closed under relative pseudo-complementation.

Definition 6. *Let K and L be prefix-closed languages. Define*

$$K \to L = \{w : w{\downarrow_n} \in K \to w{\downarrow_n} \in L \textit{ for } 0 \leq n \leq |w|\}.$$

We first show that the language $K \to L$ defined above is indeed the pseudo-complement of K relative to L.

Proposition 1. *Let K, L, and M be prefix-closed languages. Then $K \cap M \subseteq L$ if and only if $M \subseteq K \to L$.*

Next, we show that $K \to L$ is non-empty and prefix-closed if both K and L are non-empty and prefix-closed.

Lemma 3. *Let K and L be non-empty, prefix-closed languages. Then $K \to L$ is non-empty and prefix-closed.*

It remains to show that regularity is preserved by Definition 6. Given two deterministic finite state automata M and N, we construct a new automaton $M \to N$ such that $L(M \to N) = L(M) \to L(N)$. Our idea is to use an extra bit to accumulate information. Let $\mathbb{B} = \{ \textsf{false}, \textsf{true} \}$ be the Boolean domain. The following definition gives the construction of $M \to N$.

Definition 7. *Let $M = (P, p_0, \longrightarrow_M, F_M)$ and $N = (Q, q_0, \longrightarrow_N, F_N)$ be deterministic finite state automata accepting prefix-closed languages. Define the finite state automaton $M \to N = (P \times Q \times \mathbb{B}, (p_0, q_0, b_0), \longrightarrow, F)$ as follows.*

- $b_0 = \begin{cases} \textsf{true} & \textit{if } p_0 \in F_M \to q_0 \in F_N \\ \textsf{false} & \textit{otherwise} \end{cases}$

– $(p, q, b) \xrightarrow{a} (p', q', b')$ *if*
  - $p \xrightarrow{a}_M p'$;
  - $q \xrightarrow{a}_N q'$; *and*
  - $b' = \begin{cases} \mathsf{true} & \text{if } b = \mathsf{true} \text{ and } p' \in F_M \to q' \in F_N \\ \mathsf{false} & \text{otherwise} \end{cases}$

– $F = \{(p, q, \mathsf{true}) : p \in P, q \in Q\}$.

The automaton $M \to E$ of the automata M and E in Figure 4 is shown in Figure 5. Note that the bold states are the products of the unaccepting states in Figure 4. Any string prefixed by strings in $\overline{L(M)}$ and followed by those in $\overline{L(E)}$ is accepted in the accepting bold state in $M \to E$. But strings prefixed by $\overline{L(E)}$ and followed by $\overline{L(M)}$ reach the other bold state and are not accepted.

To show that $L(M \to N) = L(M) \to L(N)$, we use the following lemma.

Lemma 4. *Let $M = (P, p_0, \longrightarrow_M, F_M)$ and $N = (Q, q_0, \longrightarrow_N, F_N)$ be deterministic finite state automata accepting non-empty, prefix-closed languages. Consider any $w \in \Sigma^*$ and $(p_0, q_0, b_0) \xrightarrow{w\downarrow_n} (p_n, q_n, b_n)$ in $M \to N$ for $0 \le n \le |w|$. Then $b_{|w|}$ is true if and only if $p_n \in F_M \to q_n \in F_N$ for $0 \le n \le |w|$.*

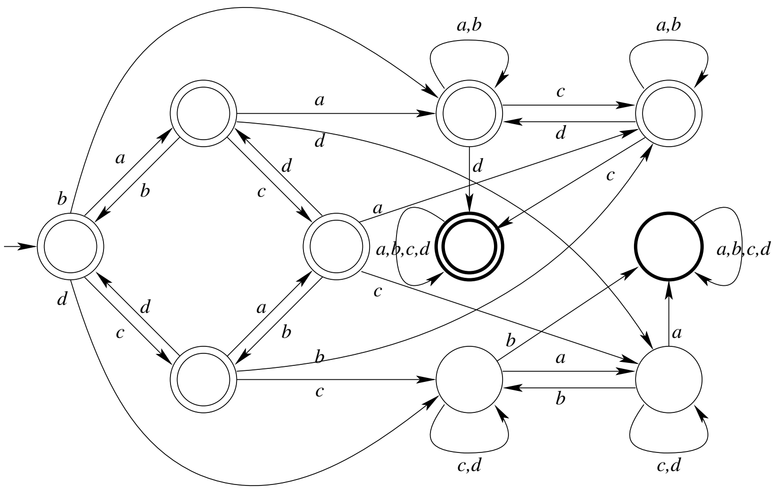


Fig. 5. The automaton $M \to E$

Now we establish $L(M \to N) = L(M) \to L(N)$ in the following proposition.

Proposition 2. *Let M and N be deterministic finite state automata accepting non-empty, prefix-closed languages. Then $L(M) \to L(N) = L(M \to N)$.*

Since any regular language is accepted by a deterministic finite state automaton, we immediately have the following proposition.

Proposition 3. *Let K and L be non-empty, prefix-closed regular languages. Then $K \to L$ is a non-empty, prefix-closed regular language.*

Define $R^+ = \{L \subseteq \Sigma^* : L$ is non-empty, prefix-closed, and regular $\}$. The following theorem states that non-empty, prefix-closed regular languages form a Heyting algebra.

Theorem 5. *Let $\mathcal{R}^+ = (R^+, \subseteq, \cup, \cap, \to, \{\epsilon\}, \Sigma^*)$. $\mathcal{R}^+$ is a Heyting algebra.*

We now turn our attention to circular compositional rules. A modified version of non-interference in [1] is used in our setting.

Definition 8. *Let L be a non-empty, prefix-closed language in Σ^* and $\Xi \subseteq \Sigma$. We say L* constrains *Ξ, write $L \rhd \Xi$, if for any $w \in L$, $wa \in L$ for any $a \notin \Xi$.*

Exploiting non-interference, we show the circular inference rule presented in Section 3.1 is sound by induction on the length of strings.

Theorem 6. *Let K and L be non-empty, prefix-closed languages such that $K \rhd \Xi_K$, $L \rhd \Xi_L$, and $\Xi_K \cap \Xi_L = \emptyset$. Then $(K \to L) \cap (L \to K) \subseteq K$.*

We can in fact characterize the language $(K \to L) \cap (L \to K)$ in Theorem 6. Note that one direction can be obtained by syntactic deduction. It is the other direction where semantic analysis is needed.

Theorem 7. *Let K and L be non-empty, prefix-closed languages such that $K \rhd \Xi_K$, $L \rhd \Xi_L$, and $\Xi_K \cap \Xi_L = \emptyset$. Then $(K \to L) \cap (L \to K) = K \cap L$.*

5 Applications

A subclass of finite state automata called labeled transition systems (LTS) is used in automated compositional reasoning [3,4]. In this section, our proof-theoretic techniques are applied to derive soundness of compositional rules for LTS's. It is noted that our formulation is equivalent to those in [4,3].

Definition 9. *Let $\Sigma_M \subseteq \Sigma$. A deterministic finite state automaton $M = (Q, q_0, \longrightarrow, F)$ is a* labeled transition system (LTS) over *Σ_M if*

1. *$q \xrightarrow{a} q$ for any $q \in Q$ and $a \in \Sigma \setminus \Sigma_M$; and*
2. *If $q_i \xrightarrow{a_{i+1}} q_{i+1}$ for $0 \leq i < n$ and $q_n \in F$ for some $n \geq 0$, then $q_i \in F$ for $0 \leq i \leq n$.*

With our formulation, it is possible to define compositions of two LTS's by product automata. Let $M = (P, p_0, \longrightarrow_M, F_M)$ and $N = (Q, q_0, \longrightarrow_N, F_N)$ be two LTS's over Σ_M and Σ_N respectively. Define the finite state automaton $M \| N = (P \times Q, (p_0, q_0), \longrightarrow, F)$ as follows.

– $(p, q) \xrightarrow{a} (p', q')$ if $p \xrightarrow{a}_M p'$ and $q \xrightarrow{a}_N q'$; and
– $F = F_M \times F_N$.

It is straightforward to verify the following proposition in our formulation.

Proposition 4. *Let $M = (P, p_0, \longrightarrow_M, F_M)$ and $N = (Q, q_0, \longrightarrow_N, F_N)$ be LTS's over Σ_M and Σ_N respectively. Then $M \| N$ is an LTS over $\Sigma_M \cup \Sigma_N$. Furthermore, $L(M\|N) = L(M) \cap L(N)$.*

Suppose LTS's M and P specify the system and the property respectively. If $L(M) \subseteq L(P)$, we say M *satisfies* P and denote it by $M \models P$.

Example 1. Let M_0, M_1, A_0, A_1, P be LTS's. Consider the following assume-guarantee rule where $\overline{M}$ denotes the complement automaton of M. Note that $\overline{M}$ is not necessarily an LTS [3].

$$\frac{M_0\|A_0 \models P \qquad M_1\|A_1 \models P \qquad L(\overline{A}_0\|\overline{A}_1) \subseteq L(P)}{M_0\|M_1 \models P}$$

By Lemma 1, Proposition 4, Theorem 1, and Theorem 4, we can establish the soundness of the rule by finding a proof tree for the following sequent.

$$M_0 \wedge A_0 \to P, M_1 \wedge A_1 \to P, \neg A_0 \wedge \neg A_1 \to P \vdash_K M_0 \wedge M_1 \to P$$

The proof tree is shown in Figure 1 (a). □

To establish circular compositional rules in our framework, the intuitionistic interpretation in Section 4 is needed. The following lemma characterizes the languages accepted by LTS's and the alphabets constrained by them.

Lemma 5. *Let $M = (Q, q_0, \longrightarrow, F)$ be an LTS over Σ_M. Then $L(M)$ is non-empty, prefix-closed, and $L(M) \rhd \Sigma_M$.*

We now prove a circular compositional rule in the following example.

Example 2. Let M_0, M_1, P_0, and P_1 be LTS's. Further, assume P_0 and P_1 are over Σ_0 and Σ_1 respectively with $\Sigma_0 \cap \Sigma_1 = \emptyset$. Consider the following circular compositional rule [2].

$$\frac{M_0\|P_1 \models P_0 \qquad P_0\|M_1 \models P_1}{M_0\|M_1 \models P_0\|P_1}$$

By Theorem 6, we have

$$\vdash_J (P_0 \to P_1) \wedge (P_1 \to P_0) \to (P_0 \wedge P_1).$$

Hence the soundness of the circular compositional rule can be established by finding a proof tree for the following sequent.

$$\begin{array}{c}(P_0 \to P_1) \wedge (P_1 \to P_0) \to (P_0 \wedge P_1), \\ M_0 \wedge P_1 \to P_0, P_0 \wedge M_1 \to P_1\end{array} \vdash_J M_0 \wedge M_1 \to P_0 \wedge P_1$$

Figure 1 (b) shows the desired proof tree. □

Proof search in Example 1 and 2 can be automated. Indeed, the proof assistant Isabelle is able to prove all rules in [3,4] with the tactic `auto ()` automatically. Meanwhile, Example 2 and intuitionistic variants of the rules in [3,4] are proved by the tactic `intuition` in Coq without human intervention.[4]

[4] More precisely, both tools use natural deduction systems equivalent to Gentzen's systems [10,11].

6 On Invertibility

We say an assume-guarantee rule is *invertible* if it is always possible to satisfy its premises when its conclusion holds.[5] Thanks to Lemma [1], we can formulate the invertibility of assume-guarantee rules as a proof search problem. However, it requires the proof systems LK and LJ. We only give an example and leave technical details in another exposition.

Example 3. Let M_0, M_1, A_0, A_1, P_0, and P_1 be LTS's. Consider the following assume-guarantee rule.

$$\frac{M_0 \| A_0 \models P_0 \quad M_1 \| A_1 \models P_1 \quad M_0 \| A_0 \models A_1 \quad M_1 \| A_1 \models A_0 \quad L(\overline{A}_0 \| \overline{A}_1) = \emptyset}{M_0 \| M_1 \models P_0 \| P_1}$$

To show the rule is invertible, it suffices to find a proof tree for the following sequent in system LK.

$$M_0 \wedge M_1 \rightarrow P_0 \wedge P_1 \vdash_K \exists A_0 A_1 . \begin{array}{c} (M_0 \wedge A_0 \rightarrow P_0) \wedge (M_1 \wedge A_1 \rightarrow P_1) \wedge \\ (M_0 \wedge A_0 \rightarrow A_1) \wedge (M_1 \wedge A_1 \rightarrow A_0) \wedge \\ (\neg A_0 \wedge \neg A_1) \leftrightarrow \mathsf{false} \end{array}$$

Isabelle can in fact find a proof for us. □

7 Conclusions

Soundness theorems for compositional reasoning rules depend on underlying computational models and can be very involved. Since it is tedious to develop new compositional rules, few such rules are available for each computational model. The situation may impede the usability of automated compositional reasoning because verifiers are forced to mould their problems in a handful of compositional rules available to them. In this paper, we apply proof theory and develop a syntactic approach to analyze compositional rules for automated compositional reasoning. With publicly available proof assistants, we are able to establish compositional rules with little human intervention.

Although all compositional rules known to us have been established automatically, it is unclear whether these proof systems are complete with respect to regular languages. It would also be of great interest if one could generate compositional rules to fit different circumstances heuristically. Moreover, although the classical interpretation can be carried over to ω-regular sets [12], it is unclear whether the intuitionistic interpretation applies as well.

Research topics combining both model checking and theorem proving are not unusual. This work may be viewed as another attempt to integrate both technologies. By exploring their theoretical foundations, another useful connection

[5] In compositional reasoning literature, the term *completeness* is often used. We adopt the proof-theoretic term here.

is found. The syntactic analysis in theorem proving can indeed automate semantic proofs of compositional reasoning in model checking. More interesting integrations of both will undoubtedly be revealed if we understand them better.

Acknowledgment. The author would like to thank Edmund Clarke, Orna Grumberg, Dale Miller, Nishant Sinha, Yih-Kuen Tsay, and anonymous reviewers for their insightful comments.

References

1. Abadi, M., Plotkin, G.D.: A logical view of composition. Theoretical Computer Science 114(1), 3–30 (1993)
2. Alur, R., Henzinger, T.: Reactive modules. Formal Methods in System Design 15(1), 7–48 (1999)
3. Barringer, H., Giannakopoulou, D., Păsăreanu, C.S.: Proof rules for automated compositional verification through learning. In: Workshop on Specification and Verification of Component-Based Systems, pp. 14–21 (2003)
4. Cobleigh, J.M., Giannakopoulou, D., Păsăreanu, C.S.: Learning assumptions for compositional verification. In: Garavel, H., Hatcliff, J. (eds.) ETAPS 2003 and TACAS 2003. LNCS, vol. 2619, pp. 331–346. Springer, Heidelberg (2003)
5. de Roever, W.P., de Boer, F., Hanneman, U., Hooman, J., Lakhnech, Y., Poel, M., Zwiers, J.: Concurrency Verification: Introduction to Compositional and Noncompositional Methods. Cambridge Tracts in Theoretical Computer Science, vol. 54. Cambridge University Press, Cambridge (2001)
6. Goldblatt, R.: Topoi: The Categorial Analysis of Logic. revised edn., Dover Publications, Mineola, NY (2006)
7. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
8. Maier, P.: Compositional circular assume-guarantee rules cannot be sound and complete. In: Gordon, A.D. (ed.) ETAPS 2003 and FOSSACS 2003. LNCS, vol. 2620, pp. 343–357. Springer, Heidelberg (2003)
9. Maier, P.: Intuitionistic LTL and a new characterization of safety and liveness. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 295–309. Springer, Heidelberg (2004)
10. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002)
11. The Coq Development Team: The Coq Proof Assistant Reference Manual: version 8.0. LogiCal Project (2004)
12. Thomas, W.: Automata on infinite objects. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 133–191. Elsevier Science Publishers, Amsterdam (1990)
13. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory. Cambridge Tracts in Theoretical Computer Science, vol. 43. Cambridge University Press, Cambridge (2000)
14. Wang, B.Y.: Automatic derivation of compositional rules in automated compositional reasoning. Technical Report TR-IIS-07-002, Institute of Information Science, Academia Sinica (2007)

Compositional Event Structure Semantics for the Internal π-Calculus[⋆]

Silvia Crafa[1], Daniele Varacca[2], and Nobuko Yoshida[3]

[1] Università di Padova
[2] PPS - Université Paris 7 & CNRS
[3] Imperial College London

Abstract. We propose the first compositional event structure semantics for a very expressive π-calculus, generalising Winskel's event structures for CCS. The π-calculus we model is the πI-calculus with recursive definitions and summations. First we model the *synchronous* calculus, introducing a notion of dynamic renaming to the standard operators on event structures. Then we model the *asynchronous* calculus, for which a new additional operator, called *rooting*, is necessary for representing causality due to new name binding. The semantics are shown to be operationally adequate and sound with respect to bisimulation.

1 Introduction

Event structures [18] are a causal model for concurrency which is particularly suited for the traditional process calculi such as CCS, CSP, SCCS and ACP. Event structures intuitively and faithfully represent *causality* and *concurrency*, simply as a partial order and an irreflexive binary relation. The key point of the generality and applicability of this model is the compositionality of the parallel composition operator: the behaviour of the parallel composition of two event structures is determined by the behaviours of the two event structures. This modularity, together with other algebraic operators such as summation, renaming and hiding, leads also to a straightforward correspondence between the event structures semantics and the operational semantics - such as the labelled transition system - of a given calculus [26].

In this paper we propose the first compositional event structure semantics of a fully concurrent variant of the π-calculus. The semantics we propose generalises Winskel's semantics of CCS [22], it is operationally adequate with respect to the standard labelled transition semantics, and consequently it is sound with respect to bisimilarity.

The π-calculus we consider is known in the literature as the πI-calculus [20], where the output of free names is disallowed. The symmetry of input and output prefixes, that are both binders, simplifies considerably the theory, while preserving most of the expressiveness of the calculi with free name passing [2,17,19].

In order to provide an event structure semantics of the π-calculus, one has in particular to be able to represent dynamic creations of new synchronisation channels, a feature

⋆ Work partially supported by MIUR (project 2005015785), Università di Padova (project CPDA061331), EPSRC (GR/S55538/01, GR/T04724/01 and GR/T03208/01) and ANR (project ParSec ANR-06-SETI-010-02).

that is not present in traditional process algebras. In Winskel's event structure semantics of CCS [22], the parallel composition is defined as product in a suitable category followed by relabelling and hiding. The product represents all conceivable synchronisations, the hiding removes synchronisations that are not allowed, while the relabelling chooses suitable names for synchronisation events. In CCS one can decide statically whether two events are allowed to synchronise, whereas in the π-calculus, a synchronisation between two events may depend on which synchronisations took place before.

Consider for instance the π-process $a(x).\overline{x}(u).\mathbf{0} \mid \overline{a}(z).z(v).\mathbf{0}$ where $a(x).P$ is an input at a, $\overline{a}(z).Q$ is an output of a new name z to a and $\mathbf{0}$ denotes the inaction. This process contains two synchronisations, first along the channel a and then along a private, newly created, channel z. The second synchronisation is possible only since the names x and z are made equal by the previous synchronisation along a. To account for this phenomenon, we define the semantics of the parallel composition by performing hiding and relabelling not uniformly on the whole event structure, but relative to the causal history of events.

The full symmetry underlying the πI-calculus theory has a further advantage: it allows a uniform treatment of causal dependencies. Causal dependencies in the π-processes arise in two ways [3,11]: by nesting prefixes (called *structural* or *prefixing* or *subject* causality) and by using a name that has been bound by a previous action (called *link* or *name* or *object* causality). While subject causality is already present in CCS, object causality is distinctive of the π-calculus. In the synchronous πI-calculus, object causality always appears under subject causality, as in $a(x).x(y).\mathbf{0}$ or in $(\boldsymbol{\nu} c)a(x).(c(z).\mathbf{0} \mid \overline{c}(w).x(y).\mathbf{0})$, where the input on x causally depends in both senses from the input on a. As a result, the causality of synchronous πI-calculus can be naturally captured by the standard prefixing operator of the event structures, as in CCS.

On the other hand, in the asynchronous πI-calculus, the bound output process is no longer a prefix: in $\overline{a}(x)P$, the continuation process P can perform any action α before the output of x on a, provided that α does not contain x. Thus the asynchronous output has a looser causal dependency. For example, in $(\boldsymbol{\nu} c)\overline{a}(x)(c(z).\mathbf{0} \mid \overline{c}(w)x(y).\mathbf{0})$, $\overline{a}(x)$ only binds the input at x, and the interaction between $c(z)$ and $\overline{c}(w)$ can perform before $\overline{a}(x)$, thus there exists no subject causality. Representing this output object causality requires a novel operator on event structures that we call *rooting*, whose construction is inspired from a recent study on Ludics [9].

In this paper we present these new constructions, and use them to obtain compositional, sound and adequate semantics for both synchronous and asynchronous πI-calculus. Proofs and more explanations can be found in the extended version [8].

2 Internal π-Calculus

This section gives basic definitions of the πI-calculus [20]. This subcalculus captures the essence of name passing with a simple labelled transition relation. In contrast with the full π-calculus, only one notion of strong bisimulation exists, and it is a congruence.

Syntax. The syntax of the monadic, synchronous πI-calculus [20] is the following, where the symbols $a, b, \ldots, x, y, z$ range over the infinite set of names denoted by $Names$.

Prefixes $\pi ::= a(x) \mid \overline{a}(x)$ *Definitions* $A(\tilde{x} \mid \mathbf{z}) = P_A$

Processes $P, Q ::= \sum_{i \in I} \pi_i.P_i \mid P \mid Q \mid (\boldsymbol{\nu} a)P \mid A\langle \tilde{x} \mid \mathbf{z} \rangle$

The syntax consists of the parallel composition, name restriction, finite summation of guarded processes and recursive definition. In $\sum_{i \in I} \pi_i.P_i$, I is a finite indexing set; when I is empty we simply write $\mathbf{0}$ and denote with $+$ the binary sum. The two prefixes $a(x)$ and $\overline{a}(x)$ represent, respectively, an input prefix and a bound output prefix. A process $a(x).P$ can perform an input at a and x is the placeholder for the name so received. The bound output case is symmetric: a process $\overline{a}(x).P$ can perform an output of the fresh name x along the channel a. Differently from the π-calculus, where both bound and free names can be sent along channels, in the πI-calculus only bound names can be communicated, modelling the so called *internal mobility*. We often omit $\mathbf{0}$ and objects (e.g. write $\overline{a}$ instead of $\overline{a}(x).\mathbf{0}$).

The choice of recursive definitions rather than replication for infinite processes is justified by the fact that the πI-calculus with replication is strictly less expressive [20]. We assume that every constant A has a unique defining equation $A(\tilde{x} \mid \mathbf{z}) = P_A$. The symbol $\tilde{x}$ denotes a tuple of distinct names, while $\mathbf{z}$ represents an infinite sequence of distinct names $\mathbb{N} \to Names$. We denote $\mathbf{z}(n)$ as z_n. The tuple $\tilde{x}$ contains all free names of P_A and the range of $\mathbf{z}$ contains all bound names of P_A. The parameter $\mathbf{z}$ does not usually appear in recursive definitions in the literature. The reason we add it is that we want to maintain the following Assumption:

Every bound name is different from any other name, either bound or free. (1)

In the π-calculus, this policy is usually implicit and maintained along the computation by dynamic α-conversion: every time the definition A is unfolded, a new copy of the process P_A is created whose bound names must be fresh. This dynamic choice of names is difficult to interpret in the event structures. Hence our recursive definitions prescribe all the names that will be possibly used for a precise semantic correspondence. Notice also that this assumption has no impact on the process behaviour since every π-process can be α-renamed so that it satisfies (1).

The set of free and bound names of P, written by $\mathrm{fn}(P)$ and $\mathrm{bn}(P)$, is defined as usual, for instance $\mathrm{fn}(\overline{a}(x).P) = \{a\} \cup (\mathrm{fn}(P) \setminus \{x\})$. As for constant processes, the definition is as follows: $\mathrm{fn}(A\langle \tilde{x} \mid \mathbf{z} \rangle) = \{\tilde{x}\}$ and $\mathrm{bn}(A\langle \tilde{x} \mid \mathbf{z} \rangle) = \mathbf{z}(\mathbb{N})$.

Operational Semantics. The operational semantics is given in the following in terms of an LTS (in late style) where we let α, β range over the set of labels $\{\tau, a(x), \overline{a}(x)\}$.

(In Late)
$$\frac{}{a(x).P \xrightarrow{a(x)} P}$$

(Out)
$$\frac{}{\overline{a}(x).P \xrightarrow{\overline{a}(x)} P}$$

(Comm)
$$\frac{P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\overline{a}(y)} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu} y)(P'\{y/x\} \mid Q')}$$

(Par)
$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

(Sum)
$$\frac{P_i \xrightarrow{\alpha} P_i'}{\sum_{i \in I} P_i \xrightarrow{\alpha} P_i'} \; i \in I$$

(Res)
$$\frac{P \xrightarrow{\alpha} P'}{(\boldsymbol{\nu} a)P \xrightarrow{\alpha} (\boldsymbol{\nu} a)P'} \; a \notin \mathrm{fn}(\alpha)$$

(REC)

$$\frac{P_A\{\tilde{y}/\tilde{x}\}\{\mathbf{w}/\mathbf{z}\} \xrightarrow{\alpha} P'}{A\langle \tilde{y} \mid \mathbf{w}\rangle \xrightarrow{\alpha} P'} \; A(\tilde{x} \mid \mathbf{z}) = P_A$$

The rules above illustrate the internal mobility characterising the πI-calculus communication. In particular, according to (COMM), we have that $a(x).P \mid \overline{a}(y).Q \xrightarrow{\tau} (\boldsymbol{\nu}y)(P\{y/x\} \mid Q)$ where the fresh name y appearing in the output is chosen as the "canonical representative" of the private value that has been communicated. In (REC), the unfolding of a new copy of the recursive process updates the sequence of bound names. The definition of the substitution $\{\mathbf{w}/\mathbf{z}\}$ can be found in [8] and is sketched in the Appendix. Note also that the use of Assumption 1, makes it unnecessary to have the side conditions that usually accompany (PAR) and (RES).

Proposition 1. *Let P be a process that satisfies Assumption 1. Suppose $P \xrightarrow{\alpha} P'$. Then P' satisfies Assumption 1.*

Example 1. Consider $A(x \mid \mathbf{z}) = x(z_0).A\langle z_0 \mid \mathbf{z}'\rangle \mid x(z_1).A\langle z_1 \mid \mathbf{z}''\rangle$, where $\mathbf{z}'(n) = \mathbf{z}(2n{+}2)$ and $\mathbf{z}''(n) = \mathbf{z}(2n{+}3)$. In this case the sequence of names $\mathbf{z}$ is partitioned into two infinite subsequences $\mathbf{z}'$ and $\mathbf{z}''$ (corresponding to even and odd name occurrences), so that the bound names used in the left branch of A are different from those used in the right branch. Intuitively $A\langle a \mid \mathbf{z}\rangle$ partially "unfolds" to $a(z_0).(z_0(z_2).A\langle z_2 \mid \mathbf{z}'_1\rangle \mid z_0(z_4).A\langle z_4 \mid \mathbf{z}'_2\rangle) \mid a(z_1).(z_1(z_3).A\langle z_3 \mid \mathbf{z}''_1\rangle \mid z_1(z_5).A\langle z_3 \mid \mathbf{z}''_2\rangle)$ with suitable $\mathbf{z}'_1, \mathbf{z}'_2, \mathbf{z}''_1, \mathbf{z}''_2$.

We end this section with the definition of strong bisimilarity in the πI-calculus.

Definition 1 (πI strong bisimilarity). *A symmetric relation $\mathcal{R}$ on πI processes is a strong bisimulation if $P \;\mathcal{R}\; Q$ implies:*

- *whenever $P \xrightarrow{\tau} P'$, there is Q' s.t. $Q \xrightarrow{\tau} Q'$ and $P'\mathcal{R}Q'$.*
- *whenever $P \xrightarrow{a(x)} P'$, there is Q' s.t. $Q \xrightarrow{a(y)} Q'$ and $P'\{z/x\}\mathcal{R}Q'\{z/y\}$.*
- *whenever $P \xrightarrow{\overline{a}(x)} P'$, there is Q' s.t. $Q \xrightarrow{\overline{a}(y)} Q'$ and $P'\{z/x\}\mathcal{R}Q'\{z/y\}$.*

with z being any fresh variable. Two processes P, Q are bisimilar*, written $P \sim Q$, if they are related by some strong bisimulation.*

This definition differs from the corresponding definition in [20] because we do not have the α-conversion rule, and thus we must allow Q to mimic P using a different bound name. The relation $\sim$ is a congruence.

3 Event Structures

This section reviews basic definitions of event structures, that will be useful in Section 4. Event structures appear in the literature in different forms, the one we introduce here is usually referred to as prime event structures [10,18,23].

Definition 2 (Event Structure). *An event structure is a triple $\mathcal{E} = \langle E, \leq, \smile \rangle$ s.t.*

- *E is a countable set of* events*;*
- *$\langle E, \leq \rangle$ is a partial order, called the* causal order*;*
- *for every $e \in E$, the set $[e) := \{e' \mid e' < e\}$, called the* enabling set *of e, is finite;*
- *$\smile$ is an irreflexive and symmetric relation, called the* conflict relation*, satisfying the following: for every $e_1, e_2, e_3 \in E$ if $e_1 \leq e_2$ and $e_1 \smile e_3$ then $e_2 \smile e_3$.*

The reflexive closure of conflict is denoted by $\asymp$. We say that the conflict $e_2 \smile e_3$ is *inherited* from the conflict $e_1 \smile e_3$, when $e_1 < e_2$. If a conflict $e_1 \smile e_2$ is not inherited from any other conflict we say that it is *immediate*. If two events are not causally related nor in conflict they are said to be *concurrent*.

Definition 3 (Labelled event structure). *Let L be a set of labels. A labelled event structure $\mathcal{E} = \langle E, \leq, \smile, \lambda \rangle$ is an event structure together with a labelling function $\lambda : E \to L$ that associates a label to each event in E.*

Intuitively, labels represent *actions*, and events should be thought of as *occurrences of actions*. Labels allow us to identify events which represent different occurrences of the same action. In addition, labels are essential when composing two event structures in a parallel composition, in that they are used to point out which events may synchronise.

In order to give the semantics of a process P as an event structure $\mathcal{E}$, we have to show how the computational steps of P are reflected into $\mathcal{E}$. This will be formalised in the Operational Adequacy Theorem 2 in Section 4, which is based on the following labelled transition systems over event structures.

Definition 4. *Let $\mathcal{E} = \langle E, \leq, \smile, \lambda \rangle$ be a labelled event structure and let e be one of its minimal events. The event structure $\mathcal{E}\lfloor e = \langle E', \leq', \smile', \lambda' \rangle$ is defined by: $E' = \{e' \in E \mid e' \not\asymp e\}$, $\leq' = \leq_{|E'}$, $\smile' = \smile_{|E'}$, and $\lambda' = \lambda_{E'}$. If $\lambda(e) = \beta$, we write $\mathcal{E} \xrightarrow{\beta} \mathcal{E}\lfloor e$.*

Roughly speaking, $\mathcal{E}\lfloor e$ is $\mathcal{E}$ minus the event e, and minus all events that are in conflict with e. The reachable LTS with initial state $\mathcal{E}$ corresponds to the computations over $\mathcal{E}$. It is usually defined using the notion of *configuration* [26]. However, by relying on the LTS as defined above, the adequacy theorem has a simpler formulation. A precise correspondence between the two notions of LTS can be easily defined.

Event structures have been shown to be the class of objects of a category [26], whose morphisms are defined as follows. Let $\mathcal{E}_1 = \langle E_1, \leq_1, \smile_1 \rangle$, $\mathcal{E}_2 = \langle E_2, \leq_2, \smile_2 \rangle$ be event structures. A *morphism* $f : \mathcal{E}_1 \to \mathcal{E}_2$ is a partial function $f : E_1 \to E_2$ such that (i) f reflects causality: if $f(e_1)$ is defined, then $\big[f(e_1)\big) \subseteq f\big([e_1)\big)$; (ii) f reflects reflexive conflict: if $f(e_1), f(e_2)$ are defined, and if $f(e_1) \asymp f(e_2)$, then $e_1 \asymp e_2$.

It is easily shown that an isomorphism in this category is a bijective function that preserves and reflects causality and conflict. In the presence of labelled event structures $\mathcal{E}_1 = \langle E_1, \leq_1, \smile_1, \lambda_1 \rangle$, $\mathcal{E}_2 = \langle E_2, \leq_2, \smile_2, \lambda_2 \rangle$ on the same set of labels L, we will consider only *label preserving* isomorphisms, i.e. isomorphisms $f : \mathcal{E}_1 \to \mathcal{E}_2$ such that $\lambda_2(f(e_1)) = \lambda_1(e_1)$. If there is an isomorphism $f : \mathcal{E}_1 \to \mathcal{E}_2$, we say that $\mathcal{E}_1, \mathcal{E}_2$ are isomorphic, written $\mathcal{E}_1 \cong \mathcal{E}_2$.

We provide here an informal description of several operations on labelled event structures, that we are going to use in the next section. See [23] for more details.

- *Prefixing* $a.\mathcal{E}$. This operation adds to the event structure a new minimal element, labelled by a, below every other event in $\mathcal{E}$. Conflict, order, and labels of original elements remain the same as in $\mathcal{E}$.
- *Prefixed sum* $\sum_{i \in I} a_i.\mathcal{E}_i$. This is obtained as the disjoint union of copies of the event structures $a_i.\mathcal{E}_i$. The order relation of the new event structure is the disjoint union of the orders of $a_i.\mathcal{E}_i$ and the labelling function is the disjoint union of the labelling functions of $a_i.\mathcal{E}_i$. As for the conflict relation, we take the disjoint union of the conflicts appearing in $a_i.\mathcal{E}_i$ and we extend it by putting in conflict every pair of events belonging to two different copies of $a_i.\mathcal{E}_i$.
- *Restriction* (or *Hiding*) $\mathcal{E} \setminus X$ where $X \subseteq L$ is a set of labels. This is obtained by removing from E all events with label in X and all events that are above (i.e., causally depend on) one of those. On the remaining events, order, conflict and labelling are unchanged.
- *Relabelling* $\mathcal{E}[f]$ where L and L' are two sets of labels and $f : L \to L'$. This operation just consists in composing the labelling function λ of $\mathcal{E}$ with the function. The new event structure is labelled over L' and its labelling function is $f \circ \lambda$.

3.1 The Parallel Composition

The parallel composition of two event structures $\mathcal{E}_1$ and $\mathcal{E}_2$ gives a new event structure $\mathcal{E}'$ whose events model the parallel occurrence of events $e_1 \in E_1$ and $e_2 \in E_2$. In particular, when the labels of e_1 and e_2 match according to an underlying synchronisation model, $\mathcal{E}'$ records (with an event $e' \in E'$) that a synchronisation between e_1 and e_2 is possible, and deals with the causal effects of such a synchronisation.

The parallel composition is defined as the categorical product followed by restriction and relabelling [26]. The categorical product is unique up to isomorphism, but it can be explicitly constructed in different ways. We give a brief outline of one such construction [10,21]. Let $\mathcal{E}_1 := \langle E_1, \leq_1, \smile_1 \rangle$ and $\mathcal{E}_2 := \langle E_2, \leq_2, \smile_2 \rangle$ be event structures. Let $E_i^* := E_i \uplus \{*\}$, where $*$ is a distinguished event. The categorical product is given by an event structure $\mathcal{E} = \langle E, \leq, \smile \rangle$ and two morphisms $\pi_i : \mathcal{E} \to \mathcal{E}_i$ (the projections). The elements of E are of the form (W, e_1, e_2) where W is a finite subset of E, and $e_i \in E_i^*$. Intuitively W is the enabling set of the event (W, e_1, e_2). Order and conflict are defined using order and conflict relations of E_1, E_2 (see [10,21] for the details). The projections are defined as $\pi_1(W, e_1, e_2) = e_1$ and $\pi_2(W, e_1, e_2) = e_2$. For event structures with labels in L, let be $L_* := L \uplus \{*\}$ where $*$ is a distinguished label. Then the labelling function of the product takes on the set $L_* \times L_*$, and we define $\lambda(W, e_1, e_2) = (\lambda_1^*(e_1), \lambda_2^*(e_2))$, where $\lambda_i^*(e_i) = \lambda_i(e_i)$ if $e_i \neq *$, and $\lambda_i^*(*) = *$.

The synchronisation model underlying the relabelling operation needed for parallel composition is formalised by the notion of *synchronisation algebra* [26]. A synchronisation algebra S is a partial binary operation $\bullet_S$ defined on L_*. If α_i are the labels of events $e_i \in E_i$, then $\alpha_1 \bullet_S \alpha_2$ is the label of the event $e' \in E'$ representing the synchronisation of e_1 and e_2. If $\alpha_1 \bullet_S \alpha_2$ is undefined, the synchronisation event is given a distinguished label bad indicating that this event is not allowed and should be deleted.

Definition 5 (Parallel Composition of Event Structures). *Let $\mathcal{E}_1, \mathcal{E}_2$ two event structures labelled over L, let S be a synchronisation algebra, and let $f_S : L_* \to L' =$*

$L_* \cup \{\mathsf{bad}\}$ be a function defined as $f_S(\alpha_1, \alpha_2) = \alpha_1 \bullet_S \alpha_2$, if S is defined on (α_1, α_2), and $f_S(\alpha_1, \alpha_2) = \mathsf{bad}$ otherwise. The parallel composition $\mathcal{E}_1 \|_S \mathcal{E}_2$ is defined as the categorical product followed by relabelling and restriction[1]*: $\mathcal{E}_1 \|_S \mathcal{E}_2 = (\mathcal{E}_1 \times \mathcal{E}_2)[f_S] \setminus \{\mathsf{bad}\}$. The subscripts S are omitted when the synchronisation algebra is clear from the context.*

Example 2. We show a simple example of parallel composition. Let $L = \{\alpha, \beta, \overline{\alpha}, \tau\}$ Consider the two event structures $\mathcal{E}_1, \mathcal{E}_2$, where $E_1 = \{a, b\}, E_2 = \{a'\}$, with $a \leq_1 b$ and $\lambda_1(a) = \alpha, \lambda_1(b) = \beta, \lambda_2(a') = \overline{\alpha}$. The event structures are represented as follows:

$$\begin{tikzcd}
 & \beta & & & & \beta & \beta & \\
\mathcal{E}_1 : & \alpha \arrow[u, dash] & \mathcal{E}_2 : & \overline{\alpha} & \mathcal{E}_3 : & \alpha \arrow[u, dash] \arrow[r, squiggly, no head] & \boldsymbol{\tau} \arrow[u, dash] \arrow[r, squiggly, no head] & \overline{\alpha}
\end{tikzcd}$$

where curly lines represent immediate conflict, while the causal order proceeds upwards along the straight lines. Consider the synchronisation algebra obtained as the symmetric closure of the following rules: $\alpha \bullet \overline{\alpha} = \boldsymbol{\tau}$, $\alpha \bullet * = \alpha$, $\overline{\alpha} \bullet * = \overline{\alpha}$, $\beta \bullet * = \beta$ and undefined otherwise. Then $\mathcal{E}_3 := \mathcal{E}_1 \| \mathcal{E}_2$ is the event structure $\langle E_3, \leq, \smile, \lambda \rangle$ where $E_3 = \{e := (\emptyset, a, *), e' := (\emptyset, *, a'), e'' := (\emptyset, a, a'), d := (\{e\}, a', *), d'' := (\{e''\}, a', *)\}$, the ordering $\leq$ is defined as $e \leq d, e'' \leq d''$, while the conflict $\smile$ is defined as $e \smile e''$, $e' \smile e''$, $e \smile d''$, $e' \smile d''$, $e'' \smile d$, $d \smile d''$. The labelling function is $\lambda(e) = \alpha$, $\lambda(e') = \overline{\alpha}, \lambda(e'') = \boldsymbol{\tau}, \lambda(d) = \lambda(d'') = \beta$.

A large CPO of event structures. We say that an event structure $\mathcal{E}$ is a *prefix* of an event structure $\mathcal{E}'$, denoted $\mathcal{E} \leq \mathcal{E}'$ if there exists $\mathcal{E}'' \cong \mathcal{E}'$ such that $E \subseteq E''$ and no event in $E'' \setminus E$ is below any event of E.

Winskel [22] has shown that the class of event structures with the prefix order is a large CPO, and thus the limits of countable increasing chains exist. Moreover all operators on event structures are continuous. We will use this fact to define the semantics of the recursive definitions.

4 Event Structure Semantics

This section defines the denotational semantics of πI-processes in terms of labelled event structures. Given a process P, we associate to P an event structure $\mathcal{E}_P$ whose events e represent the occurrence of an action $\lambda(e)$ in the LTS of P. Our main issue is compositionality: the semantics of the process $P \mid Q$ should be defined as $\mathcal{E}_P \parallel \mathcal{E}_Q$ so that the operator $\parallel$ satisfactorily models the parallel composition of P and Q.

4.1 Generalised Relabelling

It is clear from Definition 5 that the core of the parallel composition of event structures is the definition of a relabelling function encoding the intended synchronisation model.

[1] In [26], the definition of parallel composition is $(\mathcal{E}_1 \times \mathcal{E}_2 \setminus X)[f]$, where X is the set of labels (pairs) for which f is undefined. We can prove that such a definition is equivalent to ours, which is more suitable to be generalised to the π-calculus.

As discussed in the Introduction, name dependences appearing in πI-processes let a synchronisation between two events possibly depend on the previous synchronisations. We then define a generalised relabelling operation where the relabelling of an event depends on (the labels of) its causal history. Such a new operator is well-suited to encode the πI-communication model and allows the semantics of the πI-calculus to be defined as an extension of CCS event structure semantics.

Definition 6 (Generalised Relabelling). *Let L and L' be two sets of labels, and let $Pom(L')$ be a pomset (i.e., partially ordered multiset) of labels in L'. Given an event structure $\mathcal{E} = \langle E, \leq, \smile, \lambda\rangle$ over the set of labels L, and a function $f : Pom(L') \times L \longrightarrow L'$, we define the relabelling operation $\mathcal{E}[f]$ as the event structure $\mathcal{E}' = \langle E, \leq, \smile, \lambda'\rangle$ with labels in L', where $\lambda' : E \longrightarrow L'$ is defined as follows by induction on the height of an element of E:*

$$\textit{if } h(e) = 0 \textit{ then } \lambda'(e) = f(\,\emptyset,\ \lambda(e)\,)$$
$$\textit{if } h(e) = n+1 \textit{ then } \lambda'(e) = f(\,\lambda'([e)),\ \lambda(e)\,)$$

In words, an event e is relabelled with a label $\lambda'(e)$ that depends on the (pomset of) labels of the events belonging to its causal history $[e)$.

The set of labels we consider is $L = \{a(x), \overline{a}(x), \tau \mid a, x \in Names\}$. For the parallel composition we need an auxiliary set of labels $L' = \{a(x), \overline{a}(x), \tau_{x=y} \mid a, x, y \in Names\} \cup \{\mathsf{bad}, \mathsf{hide}\}$, where bad and hide are distinguished labels.

In L', the silent action τ is tagged with the couple of bound names that get identified through the synchronisation. This extra piece of information carried by τ-actions is essential in the definition of the generalised relabelling function. Let for instance e encode the parallel occurrence of two events e_1, e_2 labelled, resp., $x(x')$ and $\overline{y}(y')$, then e_1 and e_2 do synchronise only if x and y are equal, that is only if in the causal history of e there is an event labelled with $\tau_{x=y}$; in such a case e can then be labelled with $\tau_{x'=y'}$.

The distinguished label bad denotes, as before, synchronisations that are not allowed, while the new label hide denotes the hiding of newly generated names. Both labels are finally deleted.

Let $f_\pi : Pom(L') \times (L \uplus \{*\} \times L \uplus \{*\}) \longrightarrow L'$ be the relabelling function defined as:

$$f_\pi(X, \langle a(y), \overline{a}(z)\rangle) = f_\pi(X, \langle \overline{a}(z), a(y)\rangle) = \tau_{y=z}$$

$$f_\pi(X, \langle a(y), \overline{b}(z)\rangle) = f_\pi(X, \langle \overline{b}(z), a(y)\rangle) = \begin{cases} \tau_{y=z} & \text{if } \tau_{a=b} \in X \\ \mathsf{bad} & \text{otherwise} \end{cases}$$

$$f_\pi(X, \langle \alpha, *\rangle) = \quad f_\pi(X, \langle *, \alpha\rangle) \quad = \begin{cases} \mathsf{hide} & \text{if } \tau_{a=b} \in X \ \&\ \alpha = a(y), \overline{a}(y) \\ \alpha & \text{otherwise} \end{cases}$$

$$f_\pi(X, \langle \alpha, \beta\rangle) \quad = \mathsf{bad} \quad \text{otherwise}$$

The function f_π encodes the πI-synchronisation model in that it only allows synchronisations between input and output over the same channel, or over two channels whose names have been identified by a previous communication. The actions over a channel

a that has been the object of a previous synchronisation are relabelled as hide since, according to internal mobility, a is a bound name.

The extra information carried by the τ-actions is only necessary in order to *define* the relabelling, but it should later on be forgotten, as we do not distinguish τ-actions in the LTS. Hence we apply a second relabelling er that simply erases the tags:

$$er(\alpha) = \begin{cases} \tau & \text{if } \alpha = \tau_{x=y} \\ \alpha & \text{otherwise} \end{cases}$$

4.2 Definition of the Semantics

The semantics of the πI-calculus is then defined as follows by induction on processes, where the parallel composition of event structure is defined by

$$\mathcal{E}_1 \|_\pi \mathcal{E}_2 = ((\mathcal{E}_1 \times \mathcal{E}_2)\ [f_\pi][er]) \setminus \{\mathsf{bad}, \mathsf{hide}\}$$

To deal with recursive definitions, we use an index k to denote the level of unfolding.

$$\{\!|\, \mathbf{0} \,|\!\}_k = \emptyset \qquad \{\!|\, \textstyle\sum_{i \in I} \pi_i.P_i \,|\!\}_k = \sum_{i \in I} \pi_i.\{\!|\, P_i \,|\!\}_k$$

$$\{\!|\, P \mid Q \,|\!\}_k = \{\!|\, P \,|\!\}_k \|_\pi \{\!|\, Q \,|\!\}_k \qquad \{\!|\, (\nu a)P \,|\!\}_k = \{\!|\, P \,|\!\}_k \setminus \{l \in L \mid a \text{ is the subject of } l\}$$

$$\{\!|\, A\langle \tilde{y} \mid \mathbf{w} \rangle \,|\!\}_0 = \emptyset \qquad \{\!|\, A\langle \tilde{y} \mid \mathbf{w} \rangle \,|\!\}_{k+1} = \{\!|\, P_A\{\tilde{y}/\tilde{x}\}\{\mathbf{w}/\mathbf{z}\} \,|\!\}_k$$

Recall that all operators on event structures are continuous with respect to the prefix order. It is thus easy to show that, for any k, $\{\!|\, P \,|\!\}_k \leq \{\!|\, P \,|\!\}_{k+1}$. We define $\{\!|\, P \,|\!\}$ to be the limit of the increasing chain $\ldots\{\!|\, P \,|\!\}_k \leq \{\!|\, P \,|\!\}_{k+1} \leq \{\!|\, P \,|\!\}_{k+2}\ldots$:

$$\{\!|\, P \,|\!\} = \sup_{k \in \mathbb{N}} \{\!|\, P \,|\!\}_k$$

Since all operators are continuous w.r.t. the prefix order we have the following result:

Theorem 1 (Compositionality). *The semantics $\{\!|\, P \,|\!\}$ is compositional, i.e.*

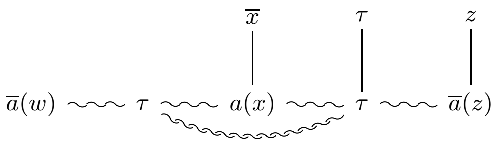
- $\{\!|\, P \mid Q \,|\!\} = \{\!|\, P \,|\!\} \|_\pi \{\!|\, Q \,|\!\}$,
- $\{\!|\, \sum_{i \in I} \pi_i.P_i \,|\!\} = \sum_{i \in I} \pi_i.\{\!|\, P_i \,|\!\}$, *and*
- $\{\!|\, (\boldsymbol{\nu} a)P \,|\!\}k = \{\!|\, P \,|\!\} \setminus \{l \in L \mid a$ *is the subject of* $l\}$.

4.3 Examples

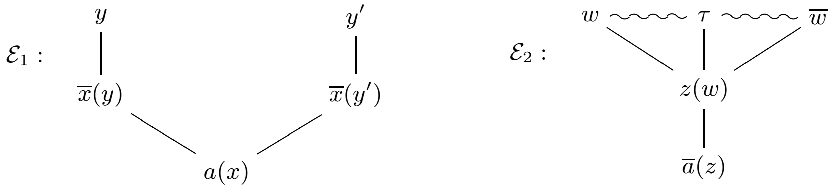
Example 3. As the first example, consider the process $P = a(x).\overline{x}(u) \mid \overline{a}(z).z(v)$ discussed in the Introduction. We show in the following the two event structures $\mathcal{E}_1, \mathcal{E}_2$ associated to the basic threads, as well as the event structure corresponding to $\{\!|\, P \,|\!\} = \mathcal{E}_1 \|_\pi \mathcal{E}_2$. Figure 1 shows two intermediate steps involved in the construction of $\{\!|\, P \,|\!\}$, according to the definition of the parallel composition operator.

$$\begin{tikzcd}
\mathcal{E}_1 : & \overline{x}(u) \arrow[d, dash] & \mathcal{E}_2 : & z(v) \arrow[d, dash] & \mathcal{E}_1 \|_\pi \mathcal{E}_2 : & \overline{x}(u) \arrow[d, dash] & \tau \arrow[d, dash] & z(v) \arrow[d, dash] \\
& a(x) & & \overline{a}(z) & & a(x) \arrow[r, dash, squiggly] & \tau \arrow[r, dash, squiggly] & \overline{a}(z)
\end{tikzcd}$$

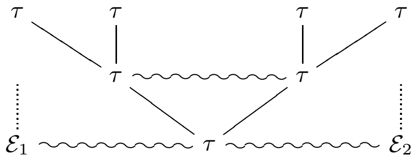
Example 4. As the second example, consider $Q = \overline{a}(w) \mid P$, where P is the process above. In Q two different communications may take place along the channel a: either the fresh name w is sent, and the resulting process is stuck, or the two threads in P can synchronise as before establishing a private channel for a subsequent communication. The behaviour of Q is illustrated by the following event structure which corresponds to $\{\!| Q |\!\} = \mathcal{E}_3 \parallel_\pi \{\!| P |\!\}$, where $\mathcal{E}_3 = \{\!| \overline{a}(w) |\!\}$ is a simple event structure consisting of a single event labeled by $\overline{a}(w)$.



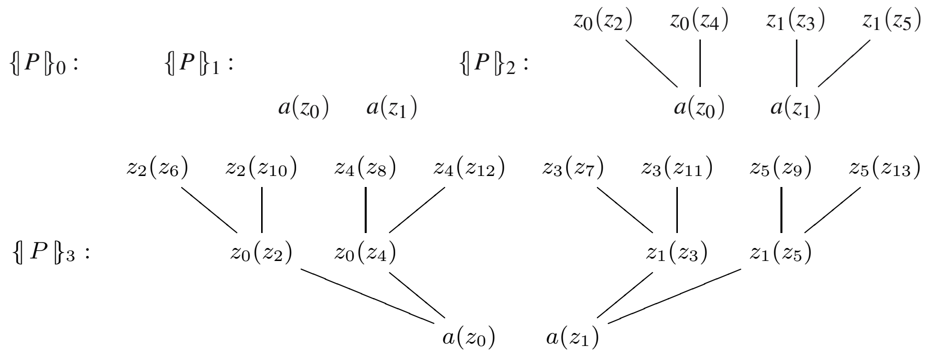
Example 5. As a further example, let $R = a(x).\big(\overline{x}(y).y \mid \overline{x}(y').y'\big) \mid \overline{a}(z).\big(z(w).(w \mid \overline{w})\big)$ whose two threads correspond to the following two event structures:



R allows a first communication on a that identifies x and z and triggers a further synchronisation with one of the outputs over x belonging to $\mathcal{E}_1$. This second communication identifies w with either y or y', which can now compete with w for the third synchronisation. The event structure corresponding to $\{\!| R |\!\} = \mathcal{E}_1 \parallel_\pi \mathcal{E}_2$ is the following.



Example 6. Consider the recursive process, seen in Example 1 in Section 2, $A(x \mid \mathbf{z}) = x(z_0).A\langle z_0 \mid \mathbf{z}'\rangle \mid x(z_1).A\langle z_1 \mid \mathbf{z}''\rangle$, where $\mathbf{z}'(n) = \mathbf{z}(2n+2)$ and $\mathbf{z}''(n) = \mathbf{z}(2n+3)$. In the following, we draw the first approximations of the semantics of $P = A\langle a \mid \mathbf{z}\rangle$:



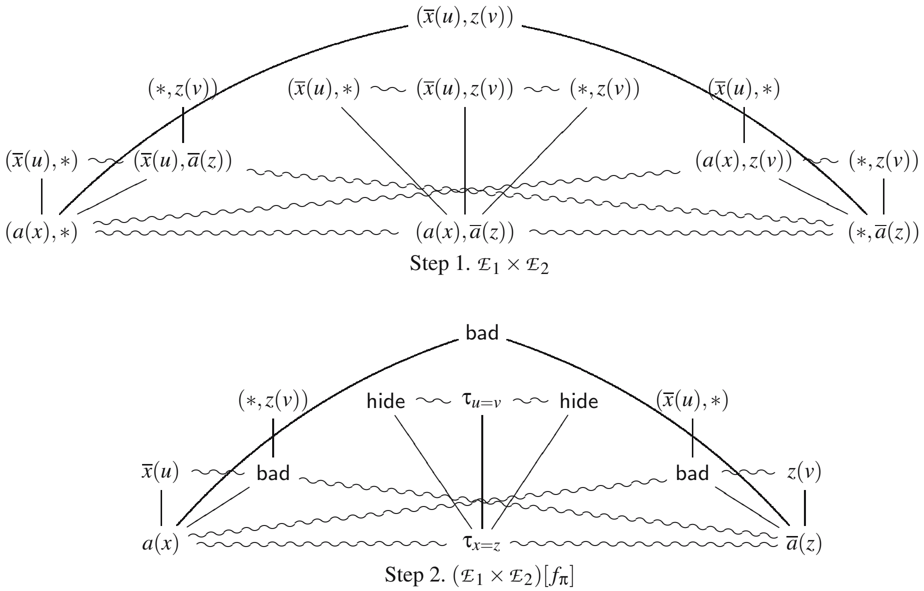


Fig. 1. Event structure corresponding to $a(x).\overline{x}(u) \mid \overline{a}(z).z(v)$

4.4 Properties of the Semantics

The operational correspondence is stated in terms of the labelled transition system defined in Section 3.

Theorem 2 (Operational Adequacy). *Suppose $P \xrightarrow{\beta} P'$ in the πI-calculus. Then $\{\!|\, P \,|\!\} \xrightarrow{\beta} \cong \{\!|\, P' \,|\!\}$. Conversely, suppose $\{\!|\, P \,|\!\} \xrightarrow{\beta} \mathcal{E}'$. Then there exists P' such that $P \xrightarrow{\beta} P'$ and $\{\!|\, P' \,|\!\} \cong \mathcal{E}'$.*

The proof technique is similar to the one used in [21], but it takes into account the generalised relabelling. As an easy corollary, we get that if two πI processes have isomorphic event structure semantics, their LTSs are isomorphic too. This clearly implies soundness w.r.t. bisimilarity.

Theorem 3 (Soundness). *If $\{\!|\, P \,|\!\} \cong \{\!|\, Q \,|\!\}$, then $P \sim Q$.*

The converse of the soundness theorem (i.e. completeness) does not hold. In fact this is always the case for event structure semantics (for instance the one in [22]), because bisimilarity abstracts away from causal relations, which are instead apparent in the event structures. As a counterexample, we have $a.b{+}b.a \;\sim\; a \mid b$ but $\{\!|\, a.b + b.a \,|\!\} \not\cong \{\!|\, a \mid b \,|\!\}$.

Isomorphism of event structures is indeed a very fine equivalence, however it is, in a sense behavioural, as it is *strictly* coarser than structural congruence.

Proposition 2. *If* $P \equiv Q$ *then* $\{\!|\, P \,|\!\} \cong \{\!|\, Q \,|\!\}$

The converse of the previous proposition does not hold: $\{\!|\, (\boldsymbol{\nu} a)a.P \,|\!\} \cong \{\!|\, \mathbf{0} \,|\!\} = \emptyset$ but $(\boldsymbol{\nu} a)a.P \not\equiv \mathbf{0}$. As a further counterexample, we have $(\boldsymbol{\nu} a)(a(x).\overline{x}(u) \mid \overline{a}(y).y(v)) \not\equiv (\boldsymbol{\nu} a, b)(a(x).\overline{b}(u) \mid \overline{a}(y).b(v))$, but both processes correspond to the same event structure containing only two events e_1, e_2 with $e_1 \leq e_2$ and $\lambda(e_1) = \lambda(e_2) = \tau$.

5 Asynchronous πI-Calculus

This section studies the *asynchronous* πI-calculus [4,15,17], whose syntax slightly differs from that in Section 2 in the treatment of the output.

$$\begin{array}{ll} \textit{Processes} & P, Q ::= \sum_{i\in I} a_i(x_i).P_i \;\mid\; \overline{a}(x)P \;\mid\; P \mid Q \;\mid\; (\boldsymbol{\nu} a)P \;\mid\; A\langle \tilde{x} \mid \mathbf{z}\rangle \\ \textit{Definition} & A(\tilde{x} \mid \mathbf{z}) = P_A \end{array}$$

The new syntax of the bound output reflects the fact that there is a looser causal connection between the output and its continuation. A process $\overline{a}(x)P$ is different from $\overline{a}(x).P$ in that it can activate the process P even if the name x has not been emitted yet along the channel a. The operational semantics can be obtained from that of Section 2 by removing the rule (OUT) and adding the following three rules:

$$\text{(OUT)}\quad \frac{}{\overline{a}(x)P \xrightarrow{\overline{a}(x)} P} \qquad \text{(ASYNC)}\quad \frac{P \xrightarrow{\alpha} P'}{\overline{a}(x)P \xrightarrow{\alpha} \overline{a}(x)P'}\; x \notin \text{fn}(\alpha) \qquad \text{(ASYNCH COMM)}\quad \frac{P \xrightarrow{a(y)} P'}{\overline{a}(x)P \xrightarrow{\tau} (\nu x)P'\{x/y\}}$$

Relying on this LTS, the definition of strong bisimilarity for the asynchronous πI-calculus is identical to that in Section 2.

5.1 Denotational Semantics

The event structure semantics of the asynchronous πI-calculus requires to encode the output process $\overline{a}(x)P$, introducing the following novel operator, called *rooting*.

Definition 7 (Rooting $a[X].\mathcal{E}$**).** *Let $\mathcal{E}$ be an event structure labelled over L, let a be a label and $X \subseteq L$ be a set of labels. We define the rooting operation $a[X].\mathcal{E}$ as the event structure $\mathcal{E}' = \langle E', \leq', \smile', \lambda'\rangle$, where $E' = E \uplus \{e'\}$ for some new event e', $\leq'$ coincides with $\leq$ on E and for every $e \in E$ such that $\lambda(e) \in X$ we have $e' \leq' e$, the conflict relation $\smile'$ coincides with $\smile$, that is e' is in conflict with no event. Finally, λ' coincides with λ on E and $\lambda'(e') = a$.*

The rooting operation adds to the event structure a new event, labeled by a, which is put below the events with labels in X (and any event above them). This operation is used to give the semantics of asynchronous bound output: given a process $\overline{a}(x)P$, every action performed by P that depends on x should be rooted with $\overline{a}(x)$. In addition to that, in order to model asynchrony, we need to also consider the possible synchronisations between $\overline{a}(x)$ and P (for example, consider $\overline{a}(x)a(z).b.x$, whose operational semantics allows an initial synchronisation between $\overline{a}(x)$ and $a(z).b.x$).

The formal construction is then obtained as follows. Given a process $\overline{a}(x)P$, every action performed by P that has x as subject is rooted with a distinctive label $\bot$. The resulting structure is composed in parallel with $\overline{a}(x)$, so that (i) every "non-blocked" action in P, (i.e. every action that does not depend on x) can synchronise with $\overline{a}(x)$, and (ii) the actions rooted by $\bot$ (i.e. those depending on x) become causally dependent on the action $\overline{a}(x)$.

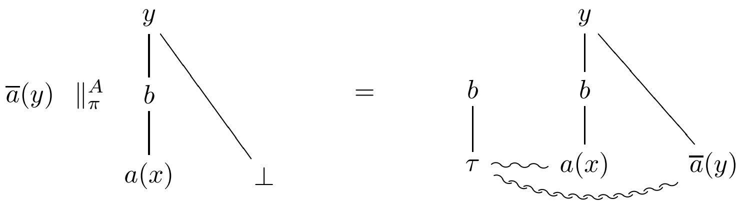
Such a composition is formalised the parallel composition operator $\|_\pi^A$ built around the generalised relabelling function $f_\pi^A : Pom(L') \times (L \uplus \{*, \bot\} \times L \uplus \{*, \bot\}) \longrightarrow L'$ that extends f_π with the following two clauses dealing with the new labels:

$$\begin{aligned} f_\pi^A(X, \langle \bot, \overline{a}(x) \rangle) &= f_\pi^A(X, \langle \overline{a}(x), \bot \rangle) = \overline{a}(x) \\ f_\pi^A(X, \langle \bot, * \rangle) &= \quad f_\pi^A(X, \langle *, \bot \rangle) = \mathsf{bad} \end{aligned}$$

The denotational semantics of asynchronous πI-processes is then identical to that in Section 4, with a new construction for the output:

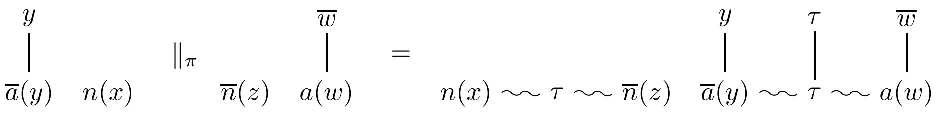
$$\{\!\!|\, \overline{a}(x)P \,|\!\!\}_k = \overline{a}(x) \ \|_\pi^A \ \bot[X].\{\!\!|\, P \,|\!\!\}_k \qquad X = \{\alpha \in L \mid x \text{ is the subject of } \alpha\}$$

Example 7. Let R be the process $\overline{a}(y)(a(x).b.y)$; its semantics is defined by the following event structure:



First a new event labelled by $\bot$ is added below any event whose label has y as subject. In this case there is only one such event, labelled by y. Then the resulting event structure is put in parallel with the single event labelled by $\overline{a}(y)$. This event can synchronise with the $\bot$ event or with the $a(x)$ event. The first synchronisation simply substitutes the label $\overline{a}(y)$ for $\bot$. The second one behaves as a standard synchronisation.

Example 8. Consider the process $P = \overline{a}(y)(n(x) \mid y) \ \mid \ \overline{n}(z)(a(w).\overline{w})$. The semantics of P is the following event structure:



Note that the causality between the $a(w)$ event and the $\overline{w}$ event is both object *and* subject, and it is due to the prefix constructor. The causality between the $\overline{a}(y)$ event and the y event is only object, and it is due to the rooting.

5.2 Properties of the Semantics

As for the synchronous case, the semantics is adequate with respect to the labelled transition system.

Theorem 4 (Operational Adequacy). *Suppose $P \xrightarrow{\beta} P'$ in the πI-calculus. Then $\{\!|\, P \,|\!\} \xrightarrow{\beta} \cong \{\!|\, P' \,|\!\}$. Conversely, suppose $\{\!|\, P \,|\!\} \xrightarrow{\beta} \mathcal{E}'$. Then there exists P' such that $P \xrightarrow{\beta} P'$ and $\{\!|\, P' \,|\!\} \cong \mathcal{E}'$.*

The proof is analogous to the synchronous case, with a case analysis for the rooting.

Theorem 5 (Soundness). *If $\{\!|\, P \,|\!\} \cong \{\!|\, Q \,|\!\}$, then $P \sim Q$.*

6 Related and Future Work

There are several causal models for the π-calculus, that use different techniques. There exist semantics in terms of labelled transition systems, where the causal relations between transitions are represented by "proofs" which allow to distinguish different occurrences of the same transition [3,11]. In [7], a more abstract approach is followed, which involves indexed transition systems. In [16], a semantics of the π-calculus in terms of pomsets is given, following ideas from dataflow theory. The two papers [6,12] present Petri nets semantics of the π-calculus.

A direct antecedent of this work presented a compositional, sound and adequate event structure semantics for a restricted, typed variant of the π-calculus [21]. This variant can embed the λ-calculus fully abstractly [1], but is strictly less expressive than the full π-calculus. The newly generated names of this subcalculus can be statically determined when typing processes, therefore the semantics presented there uses the original formulation of the parallel composition. The generalised relabelling, the rooting, and the treatment of recursive definitions are developed first in the present paper.

A recent work [5] provides an event structure semantics of the π-calculus. However this semantics does not correspond to the labelled transition semantics, but only to the *reduction* semantics, i.e. only internal silent transitions are represented in the event structure. For instance, in [5], the processes $a(x)$ and $\mathbf{0}$ have both the same semantics, the empty event structure. Consequently the semantics is neither compositional, operationally adequate, nor an extension of Winskel's semantics of CCS.

Recently Winskel [25] used event structures to give semantics to a kind of value passing CCS. His recent work [24] extends the framework of [25] to a functor category that can handle new name generation, but does not apply yet to the π-calculus.

The close relation between concurrent game semantics, linear logic and event structure semantics of the typed π-calculus has already been observed in [21,14,13]. In both worlds, the types play an important role to restrict the amount of concurrency and nondeterminism. Based on the present work, it will be interesting to extend the relation to the untyped, fully non-deterministic and concurrent framework.

Our semantics captures the essential features of the causal dependencies created by both synchronous and asynchronous name passing. For an extension of free name passing, we plan to use a technique analogous to the one developed for the asynchronous πI-calculus. As observed in [3,11], the presence of free outputs allows subtle forms of name dependences, as exemplified by $(\boldsymbol{\nu} b)(\overline{a}\langle b\rangle \mid \overline{c}\langle b\rangle)$, where a restriction contributes the object causality. A refinement of the rooting operator would be used for uniform handling name causalities induced by both internal and external mobility.

References

1. Berger, M., Honda, K., Yoshida, N.: Sequentiality and the π-calculus. In: Abramsky, S. (ed.) TLCA 2001. LNCS, vol. 2044, pp. 29–45. Springer, Heidelberg (2001)
2. Boreale, M.: On the expressiveness of internal mobility in name-passing calculi. Theor. Comp. Sci. 195(2), 205–226 (1998)
3. Boreale, M., Sangiorgi, D.: A fully abstract semantics for causality in the π-calculus. Acta Inf. 35(5), 353–400 (1998)
4. Boudol, G.: Asynchrony and the π-calculus. Research Report 1702, INRIA (1992)
5. Bruni, R., Melgratti, H., Montanari, U.: Event structure semantics for nominal calculi. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 295–309. Springer, Heidelberg (2006)
6. Busi, N., Gorrieri, R.: A petri net semantics for pi-calculus. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 145–159. Springer, Heidelberg (1995)
7. Cattani, G.L., Sewell, P.: Models for name-passing processes: Interleaving and causal. In: Proc. of LICS, pp. 322–332. IEEE Computer Society Press, Los Alamitos (2000)
8. Crafa, S., Varacca, D., Yoshida, N.: Compositional event structure semantics for the internal pi-calculus. Full version, available at www.pps.jussieu.fr/~varacca
9. Curien, P.-L., Faggian, C.: L-nets, strategies and proof-nets. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 167–183. Springer, Heidelberg (2005)
10. Degano, P., De Nicola, R., Montanari, U.: On the consistency of "truly concurrent" operational and denotational semantics. In: Proc. of LICS, pp. 133–141. IEEE Computer Society Press, Los Alamitos (1988)
11. Degano, P., Priami, C.: Non-interleaving semantics for mobile processes. Theor. Comp. Sci. 216(1-2), 237–270 (1999)
12. Engelfriet, J.: A multiset semantics for the pi-calculus with replication. Theor. Comp. Sci. 153(1&2), 65–94 (1996)
13. Faggian, C., Piccolo, M.: A graph abstract machine describing event structure composition. In: GT-VC workshop, ENTCS (2007)
14. Faggian, C., Piccolo, M.: Ludics is a model for the (finitary) linear pi-calculus. In: Proc. of TLCA. LNCS, Springer, Heidelberg (2007)
15. Honda, K., Tokoro, M.: An object calculus for asynchronous communication. In: America, P. (ed.) ECOOP 1991. LNCS, vol. 512, pp. 133–147. Springer, Heidelberg (1991)
16. Jagadeesan, L.J., Jagadeesan, R.: Causality and true concurrency: A data-flow analysis of the pi-calculus. In: Alagar, V.S., Nivat, M. (eds.) AMAST 1995. LNCS, vol. 936, pp. 277–291. Springer, Heidelberg (1995)
17. Merro, M., Sangiorgi, D.: On asynchrony in name-passing calculi. Math. Struct. Comp. Sci. 14, 715–767 (2004)
18. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part I. Theor. Comp. Sci. 13(1), 85–108 (1981)

19. Palamidessi, C.: Comparing the expressive power of the synchronous and asynchronous pi-calculi. Math. Struct. Comp. Sci. 13(5), 685–719 (2003)
20. Sangiorgi, D.: π-calculus, internal mobility and agent passing calculi. Theor. Comp. Sci. 167(2), 235–271 (1996)
21. Varacca, D., Yoshida, N.: Typed event structures and the π-calculus. In: Proc. of MFPS XXII. ENTCS, vol. 158, pp. 373–397. Elsevier, Amsterdam (2006), Full version available at `http://www.pps.jussieu.fr/~varacca`
22. Winskel, G.: Event structure semantics for CCS and related languages. In: Nielsen, M., Schmidt, E.M. (eds.) Automata, Languages, and Programming. LNCS, vol. 140, pp. 561–576. Springer, Heidelberg (1982)
23. Winskel, G.: Event structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) Advances in Petri Nets 1986. Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986. LNCS, vol. 255, pp. 325–392. Springer, Heidelberg (1987)
24. Winskel, G.: Name generation and linearity. In: Proc. of LICS, pp. 301–310. IEEE Computer Society Press, Los Alamitos (2005)
25. Winskel, G.: Relations in concurrency. In: Proc. of LICS, pp. 2–11. IEEE Computer Society Press, Los Alamitos (2005)
26. Winskel, G., Nielsen, M.: Models for concurrency. In: Handbook of logic in Computer Science, vol. 4, Clarendon Press, Oxford (1995)

A Substitution of Sequences

Let $A(\tilde{x} \mid \mathbf{z}) = P_A$ be a recursive definition. The sequence $\mathbf{z}$ contains all bound names of P_A, and in particular the names of all sequences $\mathbf{z}'$ that appear in P_A. For each such sequence, there exists an injective function $f : \mathbb{N} \to \mathbb{N}$ such that $\mathbf{z}'(n) = \mathbf{z}(f(n))$. To obtain the process $P_A\{\mathbf{w}/\mathbf{z}\}$, for each bound name of the form $\mathbf{z}(n)$ we substitute $\mathbf{w}(n)$, and for each sequence $\mathbf{z}'$ we substitute the sequence $\mathbf{w}'$ defined as $\mathbf{w}'(n) = \mathbf{w}(f(n))$.

Interpreting a Finitary Pi-calculus in Differential Interaction Nets

Thomas Ehrhard and Olivier Laurent

Preuves, Programmes & Systèmes
Université Denis Diderot and CNRS

Abstract. We propose and study a translation of a pi-calculus without sums nor replication/recursion into an untyped and essentially promotion-free version of differential interaction nets. We define a transition system of labeled processes and a transition system of labeled differential interaction nets. We prove that our translation from processes to nets is a bisimulation between these two transition systems. This shows that differential interaction nets are sufficiently expressive for representing concurrency and mobility, as formalized by the pi-calculus.

1 Introduction

Linear Logic proofs [Gir87] admit a *proof net* representation which has a very asynchronous and local reduction procedure, suggesting strong connections with parallel computation. This impression has been enforced by the introduction of *interaction nets* and *interaction combinators* by Lafont in [Laf95].

But the attempts at relating concurrency with linear logic (e.g. [EW97], [AM99], [Mel06], [Bef05], [CF06] based on [FM05]...) missed a crucial feature of true concurrency, such as modelled by process calculi like Milner's π-calculus [Mil93, SW01]: its intrinsic *non-determinism*. Indeed, all known logical systems had either an essentially deterministic reduction procedure – this is the case of intuitionistic and linear logic, and of classical systems such as Girard's LC or Parigot's $\lambda\mu$ – or an excessively non-determinitic one, as Gentzen's classical sequent calculus LK, which equates all proofs of the same formula.

However, many denotational models of the lambda-calculus and of linear logic admit some form of non-determinisms (e.g. [Plo76, Gir88]), showing that a non-deterministic proof calculus is not necessarily trivial. The first author introduced such models, based on vector spaces (see e.g. [Ehr05]), which have a nice proof-theoretic counterpart, corresponding to a simple extension of the rules that linear logic associates with the exponentials.

In this differential setting, the weakening rule has a mirror image rule called *coweakening*, and similarly for dereliction and for contraction, and the reduction rules have the corresponding mirror symmetry. The corresponding formalism of *differential interaction nets* has been introduced in a joint work by the first author and Regnier [ER06][1].

In a joint work with Kohei Honda [HL07], the second author proposed a translation of a version of the π-calculus in proof-nets for a version of linear logic extended with the cocontraction rule (as we now understand). The basic idea consists in interpreting the parallel composition as a cut between a contraction link (to which several *outputs* are connected, through dereliction links) and a cocontraction link, (to which several promoted receivers are connected.) Being promoted, these receivers are replicable, in the sense of the π-calculus. The other fundamental idea of this translation consists in using linear logic polarities for making the difference between outputs (negative) and inputs (positive), and of imposing a strict alternation between these two polarities. This allows to recast in a polarized linear logic setting a typing system for the π-calculus previously introduced by Berger, Honda and Yoshida in [BHY03]. This translation has two features which can be considered as slight defects: it accepts only replicable receivers and is not really modular (the parallel composition of two processes cannot be described as a combination of the corresponding nets).

Principle of the translation. The purpose of the present paper is to continue this line of ideas, using more systematically the new structures introduced by differential interaction nets[2].

The first key decision we made, guided by the structure of the typical cocontraction/contraction cut intended to interpret parallel composition, was of associating with each free name of a process not one, but *two* free ports in the corresponding differential interaction net. One of these ports will have a !-type (positive type) and will have to be considered as the *input port* of the corresponding name for this process, and the other one will have a ?-type (negative type) and will be considered as an *output port*.

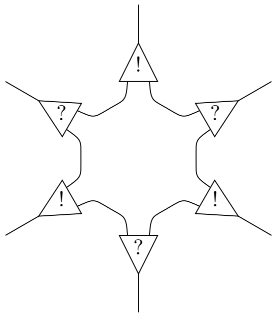


Fig. 1. Communication area

We discovered structures which allow to combine these pairs of wires for interpreting parallel composition and called them *communication areas*: they are obtained by combining in a completely symmetric way cocontraction and contraction cells. There are communication areas of any "arity" (number of pairs of

[1] Note that, in this *differential linear logic*, the two additive connectives $\oplus$ and & are identified, but this does not prevent the system from having good logical properties, and this identification – which results from non-determinism – does not extend to the multiplicative connectives: $\otimes$ and $⅋$ are distinct.

[2] One should mention here that translations of the π-calculus into nets of various kinds, subject to local reduction relations, have been provided by various authors (cf. the work of Laneve, Parrow and Victor on *solo diagrams* [LPV01], of Beffara and Maurel [BM05], of Milner on *bigraphs* [JM04], of Mazza [Maz05] on *multiport interaction nets* etc.). But these settings have no clear logical grounds nor simple denotational semantics.

wires connected to it). The communication area of arity 3 can be pictured as in Figure 1, where cocontraction cells are pictured as !-labeled triangles and contraction cells as ?-labeled triangles. The ports corresponding to the same pairs are the principal ports of antipodic cells.

Content. We first introduce differential interaction nets, typed with a recursive typing system (introduced by Danos and Regnier in [Reg92] and which corresponds to the untyped lambda-calculus) for avoiding the appearance of non reducible configurations. These nets are finitary in the sense that they use only a weak form of promotion. In this setting, we define a "toolbox", a collection of nets that we shall combine for interpreting processes, and a few associated reductions, derived from the basic reduction rules of differential interaction nets.

We organize reduction rules of nets as a labeled transition system, whose vertices are nets, and where the transitions correspond to dereliction/codereliction reduction. Then we define a process algebra which is a polyadic π-calculus, without replication and without sums. We specify the operational semantics of this calculus by means of an abstract machine inspired by the machine presented in [AC98, Chapter 16]. We define a transition system whose vertices are the states of this machine, and transitions correspond to input/output reductions. Last we define a "translation" relation from machine states to nets and show that this translation relation is a bisimulation between the two transition systems.

2 Differential Interaction Nets

Interaction nets have been introduced by Lafont [Laf95] as a generalization of linear logic proof nets. A *signature* of interaction nets is a set of *symbols*, each of them being given with an arity and a typing rule. A net is made of cells. In a net, each cell γ bears exactly one symbol, and has therefore an arity n; the cell γ must have n *auxiliary ports* (numbered from 1 to n) and one *principal port* (numbered 0). A net can also have *free ports*. Specifying the net consists last in giving its *wiring*, which is a partition of its ports in 2-elements sets (the wires). Typing the net means associating a formula of some linear logical system with each of its oriented wires in such a way that, when reversing the orientation of the wire, the formula be turned to its orthogonal. Of course, the typing rule attached to each cell of the net must also be respected by the typing.

See also [ER06] for an introduction to differential interaction nets.

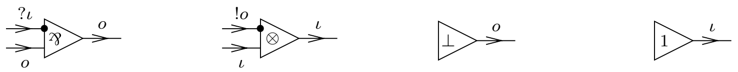
2.1 Presentation of the Cells

Our nets will be typed using a type system which corresponds to the untyped lambda-calculus. This system is based on a single type symbol o (the type of outputs), subject to the following recursive equation $o = ?o^{\perp} \parr o$. We set $\iota = o^{\perp}$, so that $\iota = !o \otimes \iota$ and $o = ?\iota \parr o$.

In the present setting, there are eleven symbols: par (arity 2), bottom (arity 0), tensor (arity 2), one (arity 0), dereliction (arity 1), weakening (arity 0), contraction (arity 2), codereliction (arity 1), coweakening (arity 0), cocontraction (arity 2) and closed promotion (arity 0). We present now the various cell symbols, with their typing rules, in a pictorial way. The principal port of a cell is located at one of the angles of the triangle representing the cell, the other ports are located on the opposite edge. We put often a black dot to locate the auxiliary port number 1.

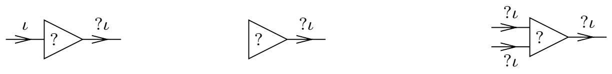
2.1.1 Multiplicative Cells

The *par* and *tensor* cells, as well as their "nullary" versions *bottom* and *one* are as follows:

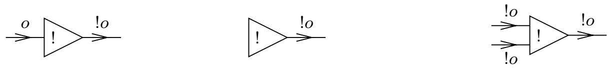


2.1.2 Exponential Cells

They are typed according to a strictly polarized discipline. Here are first the *why not* cells, which are called *dereliction*, *weakening* and *contraction*:

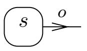
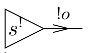


and then the *bang* cells, called *codereliction*, *coweakening* and *cocontraction*:



2.1.3 Closed Promotion Cells and the Definition of Nets

The notion of simple net is then defined inductively, together with the notion of *closed promotion* cell.

Given a (non necessarily simple) net s with only one free port  we introduce a cell .

A *simple net* is a typed interaction net, in the signature we have just defined.

A *net* is a finite formal sum of simple nets having all the same interface. Remember that the interface of a simple net s is the set of its free ports, together with the mapping associating to each free port the type of the oriented wire of s whose ending point is the corresponding port.

Let $\mathcal{L}$ be a countable set of labels containing a distinguished element τ (to be understood as the absence of label). A *labeled simple net* is a simple net where all dereliction and codereliction cells are equipped with labels belonging to $\mathcal{L}$. We require moreover that, if two labels occurring in a labeled net are equal, they are equal to τ. All the nets we consider in this paper are labeled. In our pictures, the labels of dereliction and codereliction cells will be indicated, unless it is τ, in which case the (co)dereliction cell will be drawn without any label.

3 Reduction Rules

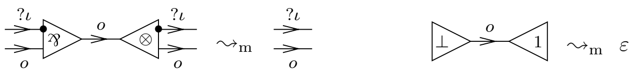
We denote by Δ the collection of all simple nets and by $\mathbb{N}\langle\Delta\rangle$ the collection of all nets (finite sums of simple nets with the same interface).

A *reduction rule* is a subset $\mathcal{R}$ of $\Delta \times \mathbb{N}\langle\Delta\rangle$ consisting of pairs (s, s') where s is made of two cells connected by their principal ports and s' has the same interface as s. This set can be finite or infinite. Such a relation is easily extended to arbitrary simple nets ($s\ \mathcal{R}\ t$ if there is $(s_0, u_1 + \cdots + u_n) \in \mathcal{R}$ where s_0 is a subnet of s, each u_i is simple and $t = t_1 + \cdots + t_n$ where t_i is obtained by replacing s_0 by u_i in s). This relation is extended to nets (sums of simple nets): $s_1 + \cdots + s_n$ (where each s_i is simple) is related to s' by this extension $\mathcal{R}^\Sigma$ if $s' = s'_1 + \cdots + s'_n$ where, for each i, $s_i\ \mathcal{R}\ s'_i$ or $s_i = s'_i$. Last, $\mathcal{R}^*$ is the transitive closure of $\mathcal{R}^\Sigma$.

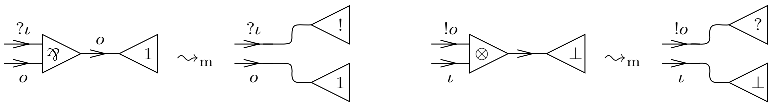
3.1 Defining the Reduction

3.1.1 Multiplicative Reduction

The first two rules concern the interaction of two multiplicative cells of the same arity.



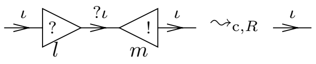
where ε stands for the empty simple net (not to be confused with the net $0 \in \mathbb{N}\langle\Delta\rangle$, the empty sum, which is not a simple net). The next two rules concern the interaction between a binary and a nullary multiplicative cell.



So here the reduction rule (denoted as $\rightsquigarrow_{\mathrm{m}}$) has four elements.

3.1.2 Communication Reduction

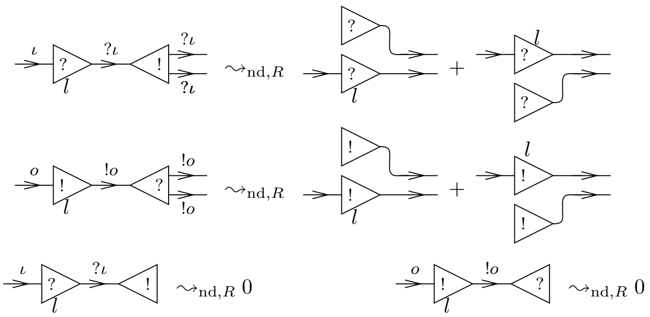
Let $R \subseteq \mathcal{L}$. We have the following reductions if $l, m \in R$.



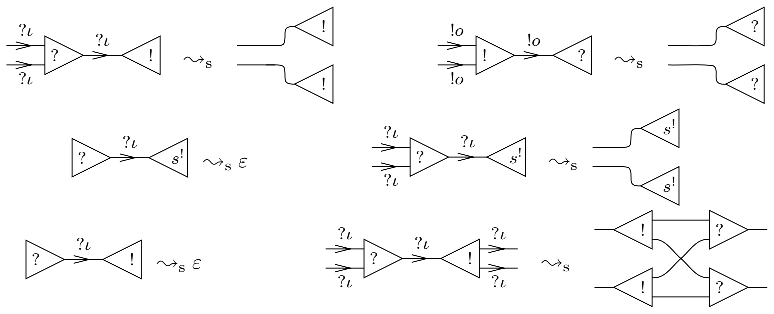
So the set $\rightsquigarrow_{\mathrm{c},R}$ is in bijective correspondence with the set of pairs (l, m) with $l, m \in R$ and $l = m \Rightarrow l = m = \tau$.

3.1.3 Non-deterministic Reduction

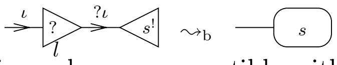
Let $R \subseteq \mathcal{L}$. We have the following reductions if $l \in R$.



3.1.4 Structural Reduction



3.1.5 Box Reduction



Observe that the reduction rules are compatible with the identification of the coweakening cell with a promotion cell containing the 0 net. Observe also that the only rules which do not admit a "symmetric" rule are those which involve a promotion cell. Indeed, promotion is the only asymmetric rule of differential linear logic.

One can check that we have provided reduction rules for all possible redexes, compatible with our typing system: for any simple net s made of two cells connected through their principal ports, there is a reduction rule whose left member is s. This rule is unique, up to the choice of a set of labels, but this choice has no influence on the right member of the rule.

3.2 Confluence

Theorem 1. *Let $R, R' \subseteq \mathcal{L}$. Let $\mathcal{R} \subseteq \Delta \times \mathbb{N}\langle\Delta\rangle$ be the union of some of the reduction relations $\rightsquigarrow_{\mathrm{c},R}$, $\rightsquigarrow_{\mathrm{nd},R'}$, $\rightsquigarrow_{\mathrm{m}}$, $\rightsquigarrow_{\mathrm{s}}$ and $\rightsquigarrow_{\mathrm{b}}$. The relation $\mathcal{R}^*$ is confluent on $\mathbb{N}\langle\Delta\rangle$.*

The proof is essentially trivial since the rewriting relation has no critical pair (see [ER06]). Given $R \subseteq \mathcal{L}$, we consider in particular the following reduction: $\rightsquigarrow_R = \rightsquigarrow_{\mathrm{m}} \cup \rightsquigarrow_{\mathrm{c},\{\tau\}} \cup \rightsquigarrow_{\mathrm{s}} \cup \rightsquigarrow_{\mathrm{b}} \cup \rightsquigarrow_{\mathrm{nd},R}$. We set $\rightsquigarrow_{\mathrm{d}} = \rightsquigarrow_{\emptyset}$ ("d" for "deterministic") and denote by $\sim_{\mathrm{d}}$ the symmetric and transitive closure of this relation.

Some of the reduction rules we have defined depend on a set of labels. This dependence is clearly monotone in the sense that the relation becomes larger when the set of labels increases.

3.3 A Transition System of Simple Nets

3.3.1 $\{l, m\}$-neutrality

Let l and m be distinct elements of $\mathcal{L} \setminus \{\tau\}$. We call (l, m)-*communication redex* a communication redex whose (co)dereliction cells are labeled by l and m. We say that a simple net s is $\{l,m\}$-*neutral* if, whenever $s \rightsquigarrow^*_{\{l,m\}} s'$, none of the simple summands of s' contains an (l,m)-communication redex.

Lemma 1. *Let s be a simple net. If $s \rightsquigarrow^*_{\{l,m\}} s'$ where all the simple summands of s' are $\{l,m\}$-neutral, then s is also $\{l,m\}$-neutral.*

3.3.2 The Transition System

We define a labeled transition system $\mathbb{D}_{\mathcal{L}}$ whose objects are simple nets, and transitions are labeled by pairs of distinct elements of $\mathcal{L} \setminus \{\tau\}$. Let s and t be simple nets, we have $s \xrightarrow{l\overline{m}} t$ if the following holds: $s \rightsquigarrow^*_{\{l,m\}} s_1 + s_2 + \cdots + s_n$ where s_1 is a simple net which contains an (l,m)-communication redex (with dereliction labeled by m and codereliction labeled by l) and becomes t when one reduces this redex, and each s_i (for $i > 1$) is $\{l,m\}$-neutral.

Lemma 2. *The relation $\sim_{\mathrm{d}} \subseteq \Delta \times \Delta$ is a strong bisimulation on $\mathbb{D}_{\mathcal{L}}$.*

4 A Toolbox for Process Calculi Interpretation

4.1 Compound Cells

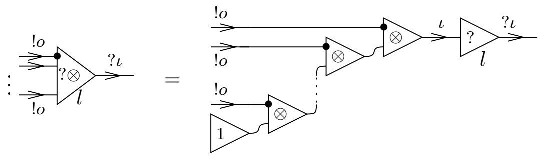
4.1.1 Generalized Contraction and Cocontraction

A *generalized contraction cell* or *contraction tree* is a simple net γ (with one principal port and a finite number of auxiliary ports) which is either a wire or a weakening cell or a contraction cell whose auxiliary ports are connected to the principal port of other contraction trees, whose auxiliary ports become the auxiliary ports of γ. Generalized cocontraction cells (cocontraction trees) are defined dually.

We use the same graphical notations for generalized (co)contraction cells as for ordinary (co)contraction cells, with a "$*$" in superscript to the "!" or "?" symbols to avoid confusions. Observe that there are infinitely many generalized (co)contraction cells of any given arity.

4.1.2 The Dereliction-Tensor and the Codereliction-Par Cells

Let n be a non-negative integer. We define an n-ary cell as follows. It will be decorated by the label of its dereliction cell (if different from τ).

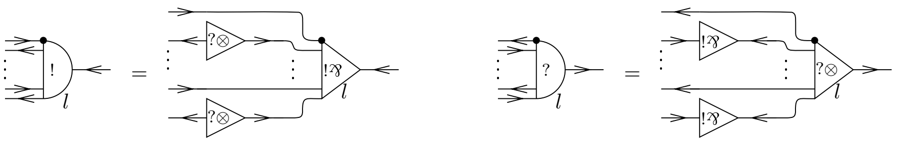


The number of tensor cells in this compound cell is equal to n. One defines dually the !⅋ compound cell.

4.1.3 The Prefix Cells

Now we can define the compound cells which will play the main role in the interpretation of prefixes of the π-calculus. Thanks to the above defined cells, all the oriented wires of the nets we shall define will bear type $?\iota$ or $!o$. Therefore we omit types and draw all wires with an orientation corresponding to the $?\iota$ type.

The *n-ary input cell* and the *n-ary output cell* are defined as



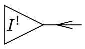
with n pairs of auxiliary ports.

Prefix cells are labeled by the label carried by their outermost dereliction-tensor or codereliction-par compound cell, if different from τ, the other codereliction-par or dereliction-tensor compound cells being unlabeled (that is, labeled by τ).

4.1.4 Transistors and Boxed Identity

In order to implement the sequentiality corresponding to sequences of prefixes in the π-calculus, we shall use the unary output prefix cell defined above as a kind of transistor, that is, as a kind of switch that one can put on a wire, and which is controlled by another wire. This idea is strongly inspired by the translation of the π-calculus in the calculus of solos[3].

These switches will be closed by "boxed identity cells", which are the unique use we make of promotion in the present work. Let I be the "identity" net of Figure 2.

Then we shall use the closed promotion cell labeled by $I^!$: .

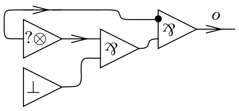


Fig. 2. Identity

4.2 Communication Tools

4.2.1 The Communication Areas

Let $n \geq -2$. We define a family of nets with $2(n+2)$ free ports, called communication areas of order n, that we shall draw using rectangles with beveled angles. Figure 3 shows how we picture a communication area of order 3.

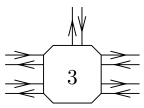
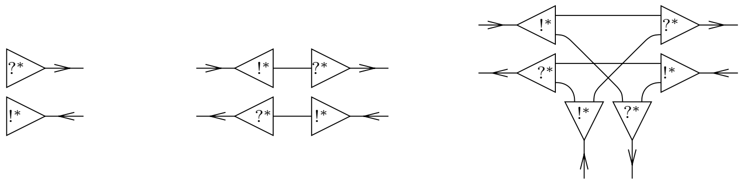


Fig. 3. Area of order 3

A communication area of order n is made of $n+2$ pairs of $(n+1)$-ary generalized cocontraction and contraction cells $(\gamma_1^+, \gamma_1^-), \ldots, (\gamma_{n+1}^+, \gamma_{n+1}^-)$, with, for each i and j such that $1 \leq i < j \leq n+2$, a wire from an auxiliary port of γ_i^+ to an auxiliary port of γ_j^- and a wire from an auxiliary port of γ_i^- to an auxiliary port of γ_j^+.

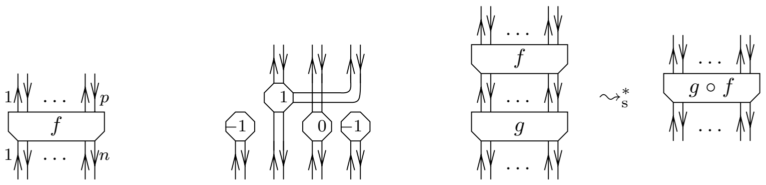
[3] It is shown in [LV03] that one can encode the π-calculus sequentiality induced by prefix nesting in the completely asynchronous solo formalism: the idea of such translations is to observe that, in a solo process like $P = \nu y\,(u(x,y) \mid y(\ldots)) \mid Q$, the first solo must interact before the second one with the environment Q.

So the communication area of order -2 is the empty net ε, and communication areas of order -1, 0 and 1 are respectively of the shape



4.2.2 Identification Structures

Let $n, p \in \mathbb{N}$ and let $f : \{1, \ldots, p\} \to \{1, \ldots, n\}$ be a function. An *f-identification* structure is a net with $p + n$ pairs of free ports (p pairs correspond to the domain of f and, in our pictures, will be attached to the non beveled side of the identification structure, and n pairs correspond to the codomain of f, attached to the beveled side of the structure) as in Figure 4(a). Such a net is made of n communication areas, and on the j'th area, the j'th pair of wires of the codomain is connected, as well as the pairs of wires of index i of the domain such that $f(i) = j$. For instance, if $n = 4$, $p = 3$, $f(1) = 2$, $f(2) = 3$ and $f(3) = 2$, a corresponding identification structure is made of four communication areas, two of order -1, one of order 0 and one of order 1, as in Figure 4(b).



(a) Notation (b) Example (c) Reduction

Fig. 4. Identification structures

4.3 Useful Reductions

4.3.1 Aggregation of Communication Areas

One of the nice properties of communication areas is that, when one connects two such areas through a pair of wires, one gets another communication area; if the two areas are of respective orders p and q, the resulting area is of order $p + q$, see Figure 5.

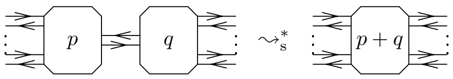


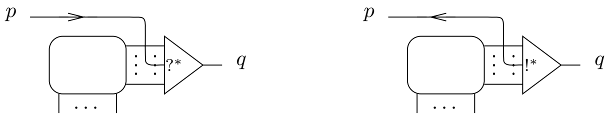
Fig. 5. Aggregation

4.3.2 Composition of Identification Structures

In particular, we get the reduction of Figure 4(c).

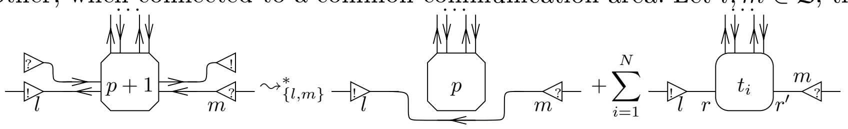
4.3.3 Port Forwarding in a Net

Let t be a net and p be a free port of t. We say that p *is forwarded in* t if there is a free port q of t such that t is of one of the two following shapes:



4.3.4 Forwarding of Derelictions and Coderelictions in Communication Areas

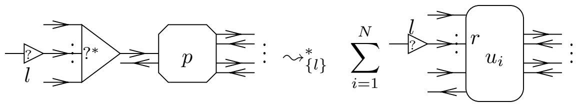
The following reduction shows that derelictions and coderelictions can meet eachother, when connected to a common communication area. Let $l, m \in \mathcal{L}$, then



where N is a non-negative integer (actually, $N = (p+1)^2$) and, in each simple net t_i, both ports r and r' are forwarded.

4.3.5 General Forwarding

Let $l \in \mathcal{L}$. The following more general but less informative property will also be used: one has



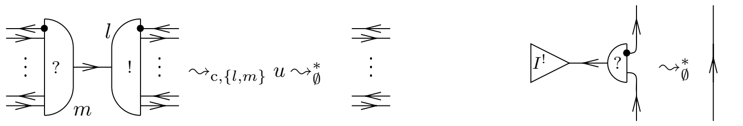
where in each simple net u_i, the port r is forwarded (see 4.3.3). Of course one also has a dual reduction (where the dereliction is replaced by a codereliction, and the generalized contraction by a generalized cocontraction).

4.3.6 Reduction of Prefixes

Let $l, m \in \mathcal{L}$. If we connect an n-ary output prefix labeled by m to a p-ary input prefix labeled by l, we obtain a net which reduces by $\rightsquigarrow_{c,\{l,m\}}$ to a net u which reduces by $\rightsquigarrow^*_{\{\emptyset\}}$ to 0 if $n \neq p$ and to simple wires, in Figure 6(a), if $n = p$.

4.3.7 Transistor Triggering

A boxed identity connected to the principal port of a unary output cell used as a "transistor" turns it into a simple wire as in Figure 6(b).



?
l
!
m
$\rightsquigarrow_{c,\{l,m\}}$ u $\rightsquigarrow^*_{\emptyset}$
I!
?
$\rightsquigarrow^*_{\emptyset}$

(a) Prefixes interaction (b) Transistor triggering

Fig. 6. Prefix reduction

5 A Polyadic Finitary π-calculus and Its Encoding

The process calculus we consider is a fragment of the π-calculus where we have suppressed the following features: sums, replication, recursive definitions, match and mismatch. This does not mean that differential interaction nets cannot interpret these features[4]. Let $\mathcal{N}$ be a countable set of names. Our processes are defined by the following syntax. We use the same set of labels as before.

- nil is the empty process.
- If P_1 and P_2 are processes, then $P_1 \mid P_2$ is a process.
- If P is a process and $a \in \mathcal{N}$, then $\nu a \cdot P$ is a process where a is bound.
- If P is a process, $a, b_1, \ldots, b_n \in \mathcal{N}$, the names b_i being pairwise distinct and if $l \in \mathcal{L}$, then $Q = [l]a(b_1 \ldots b_n) \cdot P$ is a process (prefixed by an input action, whose subject is a and whose objects are the b_is; the name a is free and each b_i is bound in Q and hence a is distinct from each b_i).
- If P is a process, $a, b_1, \ldots, b_n \in \mathcal{N}$ and $l \in \mathcal{L}$, then $\overline{[l]a}\langle b_1 \ldots b_n\rangle \cdot P$ is a process (prefixed by an output action, whose subject is a and whose objects are the b_is). This construction does not bind the names b_i, and one does not require the b_is to be distinct. The name a can be equal to some of the b_is.

The purpose of this labeling of prefixes is to distinguish the various occurrences of names as subject of prefixes. The set $\mathsf{FV}(P)$ of free names of a process P and the α-equivalence relation on processes are defined in the usual way.

A labeled process is a process where all prefixes are labeled, by pairwise distinct labels, all these labels being different from τ. If P is a labeled process, $\mathcal{L}(P)$ denotes the set of its labels. All the processes we consider in this paper are labeled.

5.1 An Execution Model

Rather than considering a rewriting relation on processes as one usually does, we prefer to define an "environment machine", similar to the machine introduced in [AC98, Chapter 16][5].

An *environment* is a function $e : \mathsf{Dom}\, e \to \mathsf{Codom}\, e$ between finite subsets of $\mathcal{N}$. A *closure* is a pair (P, e) where P is a process and e is an environment such that $\mathsf{FV}(P) \subseteq \mathsf{Dom}(e)$. A *soup* is a multiset $S = (P_1, e_1) \cdots (P_N, e_N)$ of closures (denoted by simple juxtaposition). The set $\mathsf{FV}(S)$ of free names of a soup S is the union of the codomains of the environments of S. The soup S is labeled if all the P_is are labeled, with pairwise disjoint sets of labels. A *state* is a pair

[4] Replication can be interpreted using exponential boxes, sums are probably related to the unique additive connective of differential linear logic.

[5] The reason for this choice is that the rewriting approach uses an operation which consists in replacing a name by another name in a process. The corresponding operation on nets is rather complicated and we prefer not to define it here.

(S, L) where S is a soup and L is a set of names (the names which have to be considered as local to the state) and we set $\mathsf{FV}(S, L) = \mathsf{FV}(S) \setminus L$.

The state (S, L) is labeled if the soup S is labeled. All the states we consider are labeled. One defines the set $\mathcal{L}(S, L)$ of all labels of the state (S, L) as the disjoint union of the sets of labels associated to the processes of the closures of S.

5.1.1 Canonical Form of a State

We say that a process is *guarded* if it starts with an input prefix or an output prefix. We say that a soup $S = (P_1, e_1) \cdots (P_N, e_N)$ is *canonical* if each P_i is guarded, and that a state (S, L) is canonical if the soup S is canonical. One defines a rewriting relation $\rightsquigarrow_{\mathsf{can}}$ which allows to turn a state into a canonical one.

$$\begin{aligned}
((\mathsf{nil}, e)S, L) &\rightsquigarrow_{\mathsf{can}} (S, L) \\
((\nu a \cdot P, e)S, L) &\rightsquigarrow_{\mathsf{can}} ((P, e[a \mapsto a'])S, L \cup \{a'\}) \\
((P \mid Q, e)S, L) &\rightsquigarrow_{\mathsf{can}} ((P, e)(Q, e)S, L)
\end{aligned}$$

where, in the second rule, $a' \in \mathcal{N} \setminus (L \cup \mathsf{Codom}(e) \cup \mathsf{Codom}(S))$. One shows easily that, up to α-conversion, this reduction relation is confluent, and it is clearly strongly normalizing. We denote by $\mathsf{Can}(S, L)$ the normal form of the state (S, L) for this rewriting relation. Observe that if $(S, L) \rightsquigarrow_{\mathsf{can}} (T, M)$ then $\mathsf{FV}(T, M) \subseteq \mathsf{FV}(S, L)$.

5.1.2 Transitions

Next, one defines a labeled transition system $\mathbb{S}_{\mathcal{L}}$. The objects of this system are labeled canonical states and the transitions, labeled by pairs of labels, are defined as follows.

$$\begin{aligned}
&(([l]a(b_1 \dots b_n) \cdot P, e)(\overline{[m]a'}\langle b'_1 \dots b'_n \rangle \cdot P', e')S, L) \\
&\qquad \xrightarrow{l\overline{m}} \mathsf{Can}((P, e[b_1 \mapsto e'(b'_1), \dots, b_n \mapsto e'(b'_n)])(P', e')S, L)
\end{aligned}$$

if $e(a) = e'(a')$. Observe that if $(S, L) \xrightarrow{l\overline{m}} (T, M)$ then $\mathsf{FV}(T, M) \subseteq \mathsf{FV}(S, L)$.

5.2 Translation of Processes

Since we do not work up to associativity and commutativity of contraction and cocontraction, it does not make sense to define this translation as a function from processes to nets. For each repetition-free list of names $a_1, \dots, a_n$, we define a relation $\mathcal{I}_{a_1, \dots, a_n}$ from processes whose free names are contained in $\{a_1, \dots, a_n\}$ to nets t which have $2n + 1$ free ports $a_1^\iota, a_1^o, \dots, a_n^\iota, a_n^o$ and $\mathbf{c}$ as in Figure 7(a). The additional port $\mathbf{c}$ will be used for controlling the sequentiality of the reduction, thanks to transistors. Reducing the translation of a process will be possible only when a boxed identity cell will be connected to its control port. This is completely similar to the additional control free name in the translation of the π-calculus in solos, in [LV03][6].

Clearly, if P and P' are α-equivalent, then $P \mathcal{I}_{a_1,\dots,a_n} s$ iff $P' \mathcal{I}_{a_1,\dots,a_n} s$.

5.2.1 Empty Process

One has $\mathsf{nil}\, \mathcal{I}_{b_1,\dots,b_n} t$ if t is as in Figure 7(b).

5.2.2 Name Restriction

One has $\nu a \cdot P\, \mathcal{I}_{b_1,\dots,b_n} t$ iff t is as in Figure 7(c), with s satisfying $P\, \mathcal{I}_{a,b_1,\dots,b_n} s$.

5.2.3 Parallel Composition

One has $P_1 \mid P_2\, \mathcal{I}_{b_1,\dots,b_n} t$ iff the simple net t is as in Figure 7(d), where $P_1\, \mathcal{I}_{b_1,\dots,b_n} t_1$, $P_2\, \mathcal{I}_{b_1,\dots,b_n} t_2$ and $\gamma_1,\dots,\gamma_n$ are communication areas of order 1.

5.2.4 Input Prefix

Let $l \in \mathcal{L}$. Assume that $a, b_1,\dots,b_n, c_1,\dots,c_p$ are pairwise distinct names and let $Q = [l]a(b_1 \dots b_n) \cdot P$. One has $Q\, \mathcal{I}_{a,c_1,\dots,c_p} t$ if all the free names of P are contained in $a, b_1,\dots,b_n, c_1,\dots,c_p$ and if t is as in Figure 7(e), where γ is a communication area of order 1 and where s is a simple net which satisfies $P\, \mathcal{I}_{a,b_1,\dots,b_n,c_1,\dots,c_p} s$.

5.2.5 Output Prefix

Let $l \in \mathcal{L}$. Let $b_1,\dots,b_n$ be a list of pairwise distinct names and let $Q = [l]\overline{b_{f(0)}}\langle b_{f(1)} \dots b_{f(q)}\rangle \cdot P$, where $f : \{0,1,\dots,q\} \to \{1,\dots,n\}$ is a function. One has $Q\, \mathcal{I}_{b_1,\dots,b_n} t$ if all the free names of P are contained in $b_1,\dots,b_n$ and if t is as in Figure 7(f), where $\gamma_1,\dots,\gamma_n$ are communication areas of order 1, δ is an f-identification structure and where s is a simple net which satisfies $P\, \mathcal{I}_{b_1,\dots,b_n} s$.

5.2.6 States

Let $S = (P_1, e_1) \dots (P_N, e_N)$ be a soup and $b_1,\dots,b_n$ be a repetition-free list of names containing all the codomains of the environments $e_1,\dots,e_N$. One has $S\, \mathcal{I}_{b_1,\dots,b_n} t$ if, for some simple nets s_i $(i = 1,\dots,N)$ one has $P_i\, \mathcal{I}_{b^i_1,\dots,b^i_{n_i}} s_i$ where $b^i_1,\dots,b^i_{n_i}$ is a repetition-free enumeration of the domain of e_i, and t is obtained by connecting the pair of free ports of s_i associated to each b^i_k to the corresponding pair of free port of an identification structure associated to the function e defined by $e(b^i_k) = e_i(b^i_k)$, see Figure 7(g).

[6] There is a simple interpretation of of solo diagrams into differential interaction nets, which uses only our toolbox without promotion so that solo diagrams can be seen as an intermediate graphical language which can be implemented in the low level differential syntax. Our translation of the π-calculus results from an analysis and a simplification of the composed translation "π-calculus $\to$ solo diagrams $\to$ differential nets". The simplification results from some rewiring and from the use of the boxed identity cells which is easily replicable. The translation of solos into differential nets leads to cycles (which appear when a name is identified with itself) which are avoided in the present direct translation. Well behaved conditions on solos for avoiding such cycles are introduced and studied in [EL07].

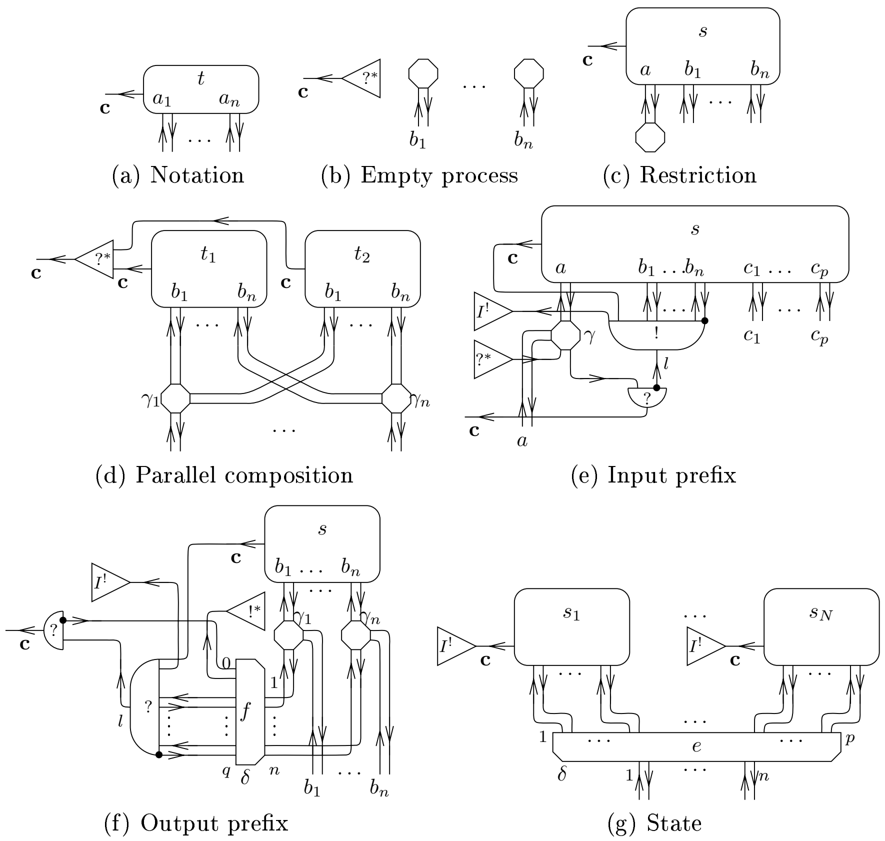


Fig. 7. Process and state translation

Last, if we are moreover given $L \subseteq \mathcal{N}$ and a repetition-free list of names $b_1, \dots, b_n$ containing all the free names of the state (S, L), one has $(S, L)\ \mathcal{I}_{b_1,\dots,b_n}\ u$ if one has $S\ \mathcal{I}_{b_1,\dots,b_n,c_1,\dots,c_p}\ t$ for some repetition-free enumeration $c_1, \dots, c_p$ of L (assumed of course to be disjoint from $b_1, \dots, b_n$) and u is obtained by plugging communication areas of order -1 on the pairs of free ports of t corresponding to the c_js.

6 Comparing the Transition Systems

We are now ready to state a bisimulation[7] theorem. Given a repetition-free list $b_1, \dots, b_n$ of names, we define a relation $\widetilde{\mathcal{I}}_{b_1,\dots,b_n}$ between states and simple nets

[7] We are not using transition systems and bisimulation in the standard process theoretic way, for analyzing the possible interactions of processes with their environment. On the contrary, we use them for describing and comparing the internal reductions of processes and nets, thanks to labels.

by: $(S, L)\ \widetilde{\mathcal{I}}_{b_1,\ldots,b_n}\ s$ if there exists a simple net s_0 such that $(S, L)\ \mathcal{I}_{b_1,\ldots,b_n}\ s_0$ and $s_0 \sim_{\mathrm{d}} s$.

Theorem 2. *The relation $\widetilde{\mathcal{I}}_{b_1,\ldots,b_n}$ is a strong bisimulation between the labeled transition systems $\mathbb{S}_{\mathcal{L}}$ and $\mathbb{D}_{\mathcal{L}}$.*

Conclusion. The main goal of this work was not to define one more translation of the π-calculus into yet another exotic formalism. We wanted to illustrate by our bisimulation result that differential interaction nets are sufficiently expressive for simulating concurrency and mobility, as formalized in the π-calculus. We believe that differential interaction nets have their own interest and find a strong mathematical and logical justification in their connection with linear logic, in the existence of various denotational models and in the analogy between its basic constructs and fundamental mathematical operations such as differentiation and convolution product. The fact that differential interaction nets support concurrency and mobility suggests that they might provide more convenient mathematical and logical foundations to concurrent computing.

References

[AC98] Amadio, R., Curien, P.-L.: Domains and lambda-calculi. Cambridge Tracts in Theoretical Computer Science, vol. 46. Cambridge University Press, Cambridge (1998)

[AM99] Abramsky, S., Melliès, P.-A.: Concurrent games and full completeness. In: Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, Los Alamitos (1999)

[Bef05] Beffara, E.: Logique, Réalisabilité et Concurrence. PhD thesis, Université Denis Diderot (2005)

[BHY03] Berger, M., Honda, K., Yoshida, N.: Strong normalisability in the pi-calculus. Information and Computation (2003) (to appear)

[BM05] Beffara, E., Maurel, F.: Concurrent nets: a study of prefixing in process calculi, vol. 356. Theoretical Computer Science (2005)

[CF06] Curien, P.-L., Faggian, C.: An approach to innocent strategies as graphs. Technical report, Preuves, Programmes et Systèmes, Submitted for publication (2006)

[Ehr05] Ehrhard, T.: Finiteness spaces. Mathematical Structures in Computer Science 15(4), 615–646 (2005)

[EL07] Ehrhard, T., Laurent, O.: Acyclic solos (submitted 2007)

[ER06] Ehrhard, T., Regnier, L.: Differential interaction nets. Theoretical Computer Science (2006) (to appear)

[EW97] Engberg, U., Winskel, G.: Completeness results for linear logic on petri nets. Annals of Pure and Applied Logic 86(2), 101–135 (1997)

[FM05] Faggian, C., Maurel, F.: Ludics nets, a game model of concurrent interaction. In: Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science, pp. 376–385. IEEE Computer Society, Los Alamitos (2005)

[Gir87] Girard, J.-Y.: Linear logic. Theoretical Computer Science 50, 1–102 (1987)

[Gir88] Girard, J.-Y.: Normal functors, power series and the λ-calculus. Annals of Pure and Applied Logic 37, 129–177 (1988)

[HL07] Honda, K., Laurent, O.: An exact correspondence between a typed π-calculus and polarized proof-nets (2007) (in preparation)

[JM04] Jensen, O., Milner, R.: Bigraphs and mobile processes (revised). Technical report, Cambridge University Computer Laboratory (2004)

[Laf95] Lafont, Y.: From proof nets to interaction nets. In: Girard, J.-Y., Lafont, Y., Regnier, L. (eds.) Advances in Linear Logic, pp. 225–247. Cambridge University Press, Cambridge (1995) Proceedings of the Workshop on Linear Logic, Ithaca, New York June 1993

[LPV01] Laneve, C., Parrow, J., Victor, B.: Solo diagrams. In: Kobayashi, N., Pierce, B.C. (eds.) TACS 2001. LNCS, vol. 2215, Springer, Heidelberg (2001)

[LV03] Laneve, C., Victor, B.: Solos in concert. Mathematical Structures in Computer Science 13(5), 657–683 (2003)

[Maz05] Mazza, D.: Multiport interaction nets and concurrency. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 21–35. Springer, Heidelberg (2005)

[Mel06] Melliès, P.-A.: Asynchronous games 2: the true concurrency of innocence. Theoretical Computer Science 358(2), 200–228 (2006)

[Mil93] Milner, R.: The polyadic pi-calculus: a tutorial. In: Logic and Algebra of Specification, pp. 203–246. Springer, Heidelberg (1993)

[Plo76] Plotkin, G.: A powerdomain construction. SIAM Journal of Computing 5(3), 452–487 (1976)

[Reg92] Regnier, L.: Lambda-Calcul et Réseaux. Thèse de doctorat, Université Paris 7 (January 1992)

[SW01] Sangiorgi, D., Walker, D.: The pi-calculus: a Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)

Mobility Control Via Passports
(Extended Abstract)

Samuel Hym

PPS, Université Paris Diderot (Paris 7) & CNRS

Abstract. Dπ is a simple distributed extension of the π-calculus in which agents are explicitly located, and may use an explicit migration construct to move between locations.

We introduce passports to control those migrations; in order to gain access to a location agents are now expected to show some credentials, granted by the destination location. Passports are tied to specific locations, from which migration is permitted. We describe a type system for these passports, which includes a novel use of dependent types, and prove that well-typing enforces the desired behaviour in migrating processes.

Passports allow locations to control incoming processes. This induces major modifications to the observations which can be made of agent-based systems. Using the type system we describe these observations, and use them to build a *loyal* notion of observational equivalence. Finally we provide a complete proof technique in the form of a bisimilarity for establishing equivalences between systems.

Keyword: process calculus; control of agent migrations; distributed computation; observational equivalence.

1 Introduction

Dπ [1] is a process calculus designed to reason about distribution of computation. It is built as a simple extension of the π-calculus in which agents are explicitly located without nesting so that a system might look like:

$$l_1[\![c\,!\,\langle b\rangle\,P_1]\!] \mid l_2[\![P_2]\!] \mid (\mathsf{new}\,a : \mathsf{E})(l_3[\![P_3]\!] \mid l_1[\![c\,?\,(x : \mathsf{T})\,P_4]\!])$$

where the l_i are location names and the P_i are processes located in one of those locations. Here, P_1 and P_4 are placed in the same location l_1, even if they are scattered in the term. Channels also are distributed: one channel is anchored in exactly one location: two processes must be in the same location to communicate. In our example, the system can evolve into

$$l_1[\![P_1]\!] \mid l_2[\![P_2]\!] \mid (\mathsf{new}\,a : \mathsf{E})(l_3[\![P_3]\!] \mid l_1[\![P_4\{^b\!/\!_x\}]\!])$$

when P_1 and P_4 communicate. This makes Dπ a streamlined distributed version of the π-calculus, which allows to concentrate our attention on agent migrations.

Dπ agents can trigger their migration from their current location, say k, to the location l via the primitive

$$\mathsf{goto}_p\, l$$

The p, added by the present work, is a *passport* which must match the actual migration attempted, from k to l. Those passports are permits, requested whenever trying to enter a location and therefore allowing that location to control which processes should be granted access.

Some other approaches to control migrations have been investigated in process calculi. In Ambients-related calculi, the migrations are particularly hard to control so many works tried to address this problem: in Safe Ambients [2], the destination location must grant access to incoming ambients by using a *co-capability*. These co-capabilities have been enriched in [3] with *passwords*: the password used to migrate is syntactically checked at *runtime* when the migration is to be granted. This idea of passwords was pursued in the NBA calculus [4] which combines it with another choice to control behaviours of ambients: communications across boundaries are allowed so that the troublesome open primitive from the original Mobile Ambients can be removed without impeding the expressivity of the calculus. This second approach was also used in different hierarchical calculi like Seal [5] or Kell [6].

In non-hierarchical calculi, we have a better handle over migrating behaviours so that more powerful techniques can be employed, for instance leveraging type systems. The present work is inspired in that direction by [7]. In that work, access to a location is a capability tied to that location via its type: access is either always granted or always denied depending on the type used when the location is generated. Of course, even when access is granted, the location name can then be transmitted without giving access; nevertheless, this setting lacks flexibility. In the present work, we refine that approach to be able to grant access selectively, depending on the origin location and to authorise such access migrations dynamically, namely after the generation of the location itself. That is why *passport* names are added to the calculus to bear those authorisations. We chose to use regular names to preserve the *homogeneity* of the calculus: in particular, they can be exchanged over channels and their scopes are dealt with in precisely the same way as any other name, including for their extrusions. Types are then used to tie rights to the names of the passports: for instance, the type $l \mapsto k$ is attached to some passport granting access to k from l. The typing system will therefore have to include dependent types to describe the link between passports and the locations they are attached to[1]. Fortunately, those dependent types bring little extra complexity to the type system itself and to the proofs of its properties, including subject reduction. What is more, this approach to tie rights to types provides type-based tools and techniques to reason about security properties. We also argue that relying on names to bear access rights gives a good handle to control those rights.

Other type systems have been used to control mobility in Dπ-based calculi. In [9], access requires the knowledge of a *port* which also governs subsequent

[1] Since a passport must grant access to only *one* location, the one which delivered that passport, using "groups" ([8]) to try and avoid dependent types would fall back on defining one group per location. So it would only reduce the expressivity of the language.

resource accesses by typing the migrating processes, using for this complex process types developed in particular in [10]. This approach is strongly constraining processes and requires higher order actions. The present work provides a first-order theory that aims at becoming a foundation for a fine-grained control of comparable power to [9]: while the passport types developed hereafter correspond to a simple mobility control, they should leave room to extensions to control resource accesses.

In [11], access to locations and resources is conditioned by policies based on the history of migrations of the agent. In the present work, the only location of the history taken into account to grant access is the origin of the migrating process: we will define a simple setting in which it is possible to describe "trust sub-networks" such as an intranet. Furthermore, the origin of a process seems easier to assert realistically than its full history. The setting we propose here relies on a simple view of trust: when a location l expresses its trust into another one k (through a passport valid from k), it also decides to trust k not to relay any dangerous process from another location.

In the following, we will investigate the notion of typed observational equivalence inherited from [12]. The founding intuition of observational equivalences is to distinguish two systems only when it is possible to observe a difference between them through a series of interactions. In a typed observational equivalence where types represents permissions, the *barbs* the observer is allowed to see are conditioned by the permissions he managed to get access to. Since permissions are represented by types, a normal type environment is used to describe the observer's rights.

Control of migrations has a great impact on the set of possible observations: since all interactions are performed over located channels, permissions to access these locations, *i.e.* passports, are mandatory to observe anything if the observer abides by the rules. We will therefore introduce an intuitive typed congruence that takes into account the migration rights of such a *loyal* observer. We argue that relying on names to bear access rights also gives a clean equivalence theory, in which the rights granted to the observer are easily expressed. As usual, the closure of the equivalence over all admissible contexts makes this equivalence intractable. So we will provide an alternative coinductive definition for this equivalence as a bisimilarity based on *actions* which identify the possible interactions between the system and its observer. This alternative definition reveals a difficulty arising from dependent types: as an artefact of dependencies, some name scopes must be opened even when the name itself is not revealed to the observer.

2 Typed Dπ with Passports

We present here a stripped-down version of the Dπ-calculus to focus on migration control. A more complete description of the specificities of the calculus we use here can be found in the long version of this work [13] or in [14]. Most of it is inherited from previous works, like [15], so we will insist mostly on the differences.

Fig. 1. Syntax for the Dπ-calculus

M ::=	*Systems*
$l[\![P]\!]$	Located process
$M_1 \mid M_2$	Parallel composition
$(\mathsf{new}\, a : \mathsf{E})\ M$	Name scope
$\mathbf{0}$	Inactive system

P ::=	*Processes*
$u\,!\,\langle V\rangle\ P$	Writing on channel
$u\,?\,(X : \mathsf{T})\ P$	Reading on channel
$\mathsf{if}\ u_1 = u_2\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$	Condition
$\mathsf{goto}_v\ u.\ P$	Migration
$\mathsf{newchan}\, c : \mathsf{C}\ \mathsf{in}\ P$	Channel generation
$\mathsf{newloc}\, l, (\vec{c}), (\vec{p}), (\vec{q}) : \mathsf{L}\ \mathsf{with}\ P_l\ \mathsf{in}\ P$	Location generation
$\mathsf{newpass}\, p\ \mathsf{from}\ \tilde{u}^\star\ \mathsf{in}\ P$	Passport generation
$P_1 \mid P_2$	Parallel composition
$*P$	Replication
stop	Termination

Processes are described using *names* (usually written $a, b, \ldots$, reserving c, d for *channel* names, k, l for *locations* and p, q for *passports*) and *variables* (usually written $x, y, \ldots$). When both names and variables can be used, we will talk of *identifiers* and write them $u, v, \ldots$. We will write $\tilde{u}$ for a set of identifiers and $\vec{u}$ for a tuple. We will also write $\tilde{u}^\star$ when either $\tilde{u}$ or $\star$ is expected (the meaning of $\star$ will be explained shortly). Finally, we will use capital letters when tuples are allowed so V can represent $(v_1, (v_2, v_3), v_4)$ or any other *value*, composed of identifiers and X any *pattern*, composed of variables.

The syntax of Dπ is given in Figure 1. Our contributions are:

– The migration construct $\mathsf{goto}_v\ u$ now mentions the *passport* v to get access to the location u.
– The new construct to generate passports, newpass, provides two kinds of origin control:
  - passports that allow migration from a given set of originating locations $\tilde{u}$ are created by $\mathsf{newpass}\, p\ \mathsf{from}\ \tilde{u}$; thus a location can express its trust in the sub-network $\tilde{u}$: every process using p will be granted access from any location in $\tilde{u}$;
  - *universal passports*, that allow migration from any location (for instance when describing the behaviour of a public server accepting requests from anywhere) are created by $\mathsf{newpass}\, p\ \mathsf{from}\ \star$.

  Of course, the location a passport grants access to is the location where the passport is generated: that is the only way to allow locations to control incoming processes.
– The construct to generate new locations, newloc, is enriched: passports to access the new location (*child*) or the location where the construct is called (*mother*) can be generated on the fly. This is the only way to model all the

Fig. 2. Reduction semantics, extracts

$$\text{(R-GOTO)}\quad l[\![\mathsf{goto}_p\, k.\ P]\!] \longrightarrow k[\![P]\!]$$

$$\text{(R-NEWLOC)}\quad l[\![\mathsf{newloc}\, k, (\vec{c}), (\vec{p}), (\vec{q}) : \textstyle\sum x : \text{LOC}.\ \mathsf{T}\ \mathsf{with}\ P_k\ \mathsf{in}\ P]\!]$$
$$\longrightarrow (\mathsf{new}\langle k, ((\vec{c}), (\vec{p}), (\vec{q})) : \mathsf{T}\{^l\!/\!_x\}\rangle)(k[\![P_k]\!] \mid l[\![P]\!])$$

$$\text{(R-NEWPASS)}\quad l[\![\mathsf{newpass}\, p\ \mathsf{from}\ \tilde{k}^\star\ \mathsf{in}\ P]\!] \longrightarrow (\mathsf{new}\, p : \tilde{k}^\star \mapsto l)\, l[\![P]\!]$$

$$\text{(R-COMM)}\quad l[\![a\,!\,\langle V\rangle\, P_1]\!] \mid l[\![a\,?\,(X : \mathsf{T})\, P_2]\!] \longrightarrow l[\![P_1]\!] \mid l[\![P_2\{^V\!/\!_X\}]\!]$$

possible situations (the child location granting access to processes from the mother location; or vice versa; and any other variation). Indeed, if passports to access the child were always created from inside the child itself, some passports granting access from the child would be needed to export them…

Since passports allow a location to accept processes depending on their origin, they can be delivered for specific communications, for instance the response awaited from a server: in

$$\begin{aligned}&cl[\![\mathsf{newpass}\, pass\ \mathsf{from}\ sv\ \mathsf{in}\\ &\qquad\qquad \mathsf{goto}_{p_{sv}}\, sv.\ req\,!\,\langle (sv, cl), (quest, res, pass)\rangle \mid \ldots res\,?\,(x)\, P]\!]\end{aligned}$$

the client *cl* generates a passport specific to the server *sv* before going there and requesting some computation while waiting for the result in *cl*. The corresponding server might look like:

$$sv[\![*req\,?\,((x_{sv}, x_{cl}), (x_{quest}, x_{res}, x_{pass}) : \mathsf{T}) \cdots \mathsf{goto}_{x_{pass}}\, x_{cl}.\ x_{res}\,!\,\langle r\rangle]\!]$$

Let us consider now the semantics associated with the calculus: the most interesting rules concerning this work are given in Figure 2 (the full set is provided in appendix). Those rules are fairly unsurprising since passports are homogeneously added to the calculus. In the reduction rule for the migration (R-GOTO), the passport involved is simply ignored: the verification of the passport will be performed using *types*. In the two rules for generation, types are instantiated in a similar way to what is usually done for channels: when passports are actually generated in (R-NEWPASS), they are tied to the location to which they will grant access, by getting the type $\tilde{k} \mapsto l$ (from $\tilde{k}$ to l).

The types we can associate with identifiers or values are summed up in Figure 3. Two major modifications are made here. Firstly, we introduce a new type for passport: $\tilde{u} \mapsto v$ will be the type of a passport to access v from one of the locations in $\tilde{u}$ and $\star \mapsto v$ of a universal passport to v. Secondly, we add a dependent sum type for values that are transmitted over channels: since the type for a passport mentions the names of the source and target locations, the dependent sum provides a way to send those names (locations and passport), packed together. They are also used to describe the tie between the locations and the passports in the newloc construct. So that the system

$$l[\![\mathsf{newloc}\, k, p, q : \sum x : \text{LOC}.\ \sum y : \text{LOC}.\ x \mapsto y, y \mapsto x\ \mathsf{with}\ P_k\ \mathsf{in}\ P]\!]$$

Fig. 3. Syntax of types

$\mathsf{E} ::=$	*Identifiers types*
$\textsc{loc}$	Location
$\textsc{r}\langle \mathsf{T}_1 \rangle_{@} u$	Channel in location u: right to read values of type T_1
$\textsc{w}\langle \mathsf{T}_2 \rangle_{@} u$	Channel in location u: right to write values of type T_2
$\textsc{rw}\langle \mathsf{T}_1, \mathsf{T}_2 \rangle_{@} u$	Intersection of the two previous types
$\tilde{u}^\star \mapsto v$	Passport
$\mathsf{T} ::=$	*Transmissible values types*
E	Identifier
$(\mathsf{T}_1, \ldots, \mathsf{T}_n)$	Tuple
$\sum \vec{x} : \vec{\textsc{loc}}.\ \mathsf{T}$	Dependent sum
$\mathsf{L} ::=$	*Types to declare locations*
$\sum x : \textsc{loc}.\ \sum y : \textsc{loc}.\ (\mathsf{C}_{1@} y, \ldots), (\tilde{u}_1^\star \mapsto y, \ldots), (\tilde{v}_1^\star \mapsto x, \ldots)$	

reduces into

$$(\mathsf{new}\, k : \textsc{loc})\, (\mathsf{new}\, p : l \mapsto k)\, (\mathsf{new}\, q : k \mapsto l)\, k[\![P_k]\!] \mid l[\![P]\!]$$

For space reasons we refer the reader to the long version [13] for the full explanations of the L types, in particular their unwinding into simple types for every identifiers, as they bring little insight on the actual passports.

Again, we provide here only a simple presentation of the set of types to focus on passports (in particular, we got rid of the recursive types which are completely orthogonal to passports types; see [16] for a detailed account of recursive types). The main property of interest about passport types is subtyping: for instance, a universal passport to access l allows to come from anywhere so should be a subtype of any passport to l. The following inference rules sum up subtyping for passports:

$$(\textsc{sr-pass})\quad \frac{\tilde{u}' \subseteq \tilde{u}}{\tilde{u} \mapsto v <: \tilde{u}' \mapsto v} \qquad (\textsc{sr-pass-}^*)\quad \star \mapsto v <: \tilde{u}^\star \mapsto v$$

We refer the reader to previous works (in particular [16]) for a complete presentation of subtyping in Dπ. Let us simply state here that the property of partial meets is preserved in this setting:

Theorem 1 (Partial meets). *Any two types sharing a subtype have a meet.*

As usual, the type system relies on *type environments*, written Γ, Φ, Ω, which are lists of hypotheses, *i.e.* associations of types to identifiers, for instance $l : \textsc{loc}, k : \textsc{loc}, p : \star \mapsto k, \ldots$ Those environments are used to prove typing judgements like $\Gamma \vdash p : l \mapsto k$, which states that p can be used to migrate from l to k according to Γ, or $\Gamma \vdash_l P$, which states that running the process P in location l

will require at most the permissions contained in Γ. These judgments are derived using inference rules: we give here only some rules relevant to passports.

$$\text{(T-GOTO)}\ \frac{\Gamma \vdash u : w \mapsto v \quad \Gamma \vdash_v P}{\Gamma \vdash_w \mathsf{goto}_u\, v.\ P} \qquad \text{(T-NEWPASS)}\ \frac{\Gamma \vdash \tilde{u} : \widetilde{\text{LOC}} \quad \Gamma; p : \tilde{u}^\star \mapsto w \vdash_w P}{\Gamma \vdash_w \mathsf{newpass}\, p\, \mathsf{from}\, \tilde{u}^\star\, \mathsf{in}\, P}$$

Those rules are fairly straightforward and provide the two expected theorems about the type system: subject reduction and type safety. Let us state here only the important part for passports using an *erroneous reduction* of a system M, written $M \xrightarrow{\text{err}}_\Gamma$, defined by the reduction $l[\![\mathsf{goto}_p\, k.\ P]\!] \xrightarrow{\text{err}}_\Gamma$ in any context whenever $\Gamma \not\vdash p : l \mapsto k$. A really simple case of erroneous reduction might look like this (this reduction is erroneous in any well-formed environment, in particular the empty one):

$$(\mathsf{new}\, l_1, l_2, l_3 : \text{LOC})\, (\mathsf{new}\, p : l_1 \mapsto l_2)\, l_1[\![\mathsf{goto}_p\, l_3.\ \mathbf{0}]\!] \xrightarrow{\text{err}}_\emptyset$$

Theorem 2. *$\Gamma \vdash M$ and $M \longrightarrow^* N$ imply $N \overset{err}{\not\longrightarrow}_\Gamma$.*

3 Loyal Observational Equivalence

The main goal of passports is to allow a location to control the processes it accepts. Naturally, this implies that the *observable* behaviour of a system depends on the actual authorisations the *observer* is granted. Let us then define an equivalence that takes passports into account drawing inspiration from [17].

For this, we will describe explicitly the knowledge of the observer, including his passports, using a type environment written Ω. This type environment thus describes the observations that can be performed, in a similar way to the knowledge-indexed relations defined in [7]. The basic observations must be interactions with the studied system, *i.e.* communications over some channels. Since channels are located, this will be possible only when the observer is granted access to their location. To actually allow the system to "choose" which locations should be reachable, we decided to place the observer into a *fresh* location. This implies that the only *directly reachable* locations are the destinations of the *universal passports* in Ω. So we define *barbs* thus:

Definition 1 (Barbs). *M shows a barb on c to Ω, written $\Omega \rhd M \Downarrow c$, whenever there exist a location l and a passport p such that: $\Omega \vdash p : \star \mapsto l$; $\Omega \vdash c : \text{R}\langle \mathsf{T} \rangle @ l$, for some type T; and there exist some P, M' and $(\vec{a} : \vec{\mathsf{E}})$ with $c, l \notin \vec{a}$ and such that $M \longrightarrow^* \equiv (\mathsf{new}\, \vec{a} : \vec{\mathsf{E}})(M' \mid l[\![c!\langle V \rangle\, P]\!])$.*

Note that the only control performed in this definition is whether the observer is able to reach the location where the interaction takes place: since our mobility control happens only when *entering* a location, it will always be possible to report the observation in the observer's home location.

Some observer knowing Ω will be able to distinguish two systems as soon as they show different sets of barbs. To get an equivalence out of this simple property, the observer is usually allowed to test the system by putting it in any context in order to eventually obtain a distinguishing barb. In our setting, we should consider only *loyal* contexts, *i.e.* contexts which use only rights available to the observer: they should not try to launch code in *unreachable* locations and access channels without the corresponding permissions. We formally define a location l as *reachable* knowing Ω when there exist $p : \star \mapsto l_1$, $p_1 : l_1 \mapsto l_2$, $\ldots$, $p_n : l_n \mapsto l$ in Ω. We will write $\mathcal{R}_\Omega$ for the set of such reachable locations. Then a context of the form $[\cdot] \mid l[\![P]\!]$ is *loyal* only when l is reachable and P is well-typed in Ω. The observer must also be *loyal* when introducing new names (for instance to be used in P):

Definition 2 (Loyal extension). *Γ' is a* loyal extension *of Γ when:*

- *$\Gamma; \Gamma'$ is a well-formed environment;*
- *for every $u : \mathsf{C}_{@}w$ when $w : \textsc{loc} \in \Gamma$, we must have $w \in \mathcal{R}_\Gamma$;*
- *for every $u : \tilde{v}^\star \mapsto w \in \Gamma'$ when $w : \textsc{loc} \in \Gamma$, we must have $w \in \mathcal{R}_\Gamma$.*

Finally, we define the loyal contextuality of a relation $\mathcal{S}$. writing $\Omega \vDash M \;\mathcal{S}\; N$ when M and N are in $\mathcal{S}$ for an observer knowing Ω. For this we need to extend subtyping to environments: we will say that Γ is a subtype of Γ' as soon as every typing judgment that can be inferred in Γ' can also be inferred in Γ.

Definition 3 (Loyally contextual relation). *A relation $\mathcal{S}$ is said* loyally contextual *only when:*

- *If $\Omega \vDash M \;\mathcal{S}\; N$ and Ω' is a loyal extension of Ω such that for every $a : \mathsf{E}$ in Ω', a is fresh, then $\Omega; \Omega' \vDash M \;\mathcal{S}\; N$.*
- *If $\Omega \vDash M \;\mathcal{S}\; N$, $k \in \mathcal{R}_\Omega$ and $\Omega \vdash k[\![P]\!]$ then $\Omega \vDash M \mid k[\![P]\!] \;\mathcal{S}\; N \mid k[\![P]\!]$.*
- *If $\Omega; a : \mathsf{E} \vDash M \;\mathcal{S}\; N$ and both $(\mathsf{new}\, a : \mathsf{E})\, M$ and $(\mathsf{new}\, a : \mathsf{E})\, N$ are well-typed in some subtype environment of Ω, then $\Omega \vDash (\mathsf{new}\, a : \mathsf{E})\, M \;\mathcal{S}\; (\mathsf{new}\, a : \mathsf{E})\, N$.*

The loyal barbed congruence follows from the notion of contextuality.

Definition 4 (Loyal barbed congruence). *We call* loyal barbed congruence*, written $\cong^l$, the biggest symmetric loyally contextual relation that preserves barbs and is closed over reductions.*

The contexts considered in this congruence can launch processes in every *reachable* location (to allow more contexts to be used) while barbs can only be observed in *directly reachable* (to get the simplest notion of observations). Note though that the congruence obtained does not depend on this choice because it is closed: it is simple to see that *reachable barbs* or *directly reachable contexts* would end up defining the same equivalence.

4 Loyal Bisimilarity

The definition given for the loyal barbed congruence is justified by intuitions but it is highly intractable: every proof of equivalence indeed requires a quantification over all contexts. So we also propose a complete proof technique for

Fig. 4. Labelled transition system, most significant rules

$$\text{(LTS-GOTO)} \quad \Omega \triangleright l[\![\mathsf{goto}_p\, k.\ P]\!] \xrightarrow{\tau} \Omega \triangleright k[\![P]\!]$$

$$\text{(LTS-W)} \quad \frac{l \in \mathcal{R}_\Omega \qquad \Omega \vdash_l a : \mathrm{R}\langle \mathsf{T} \rangle \quad \text{where } \mathsf{T} = \Omega^r(a)}{\Omega \triangleright l[\![a\,!\,\langle V \rangle\, P]\!] \xrightarrow{a!V} \Omega, \langle V : \mathsf{T} \rangle \triangleright l[\![P]\!]}$$

$$\text{(LTS-R)} \quad \frac{l \in \mathcal{R}_\Omega \qquad \Omega \vdash_l a : \mathrm{W}\langle \mathsf{T}' \rangle \qquad \Omega \vdash_l V : \mathsf{T}'}{\Omega \triangleright l[\![a\,?\,(X : \mathsf{T})\, P]\!] \xrightarrow{a?V} \Omega \triangleright l[\![P\{^V\!/x\}]\!]}$$

$$\text{(LTS-COMM)} \quad \frac{\begin{array}{c}\Omega_M \triangleright M \xrightarrow{(\Phi)a!V} \Omega'_M \triangleright M' \\ \Omega_N \triangleright N \xrightarrow{(\Phi)a?V} \Omega'_N \triangleright N'\end{array}}{\begin{array}{c}\Omega \triangleright M \mid N \xrightarrow{\tau} \Omega \triangleright (\mathsf{new}\,\Phi)\, M' \mid N' \\ \Omega \triangleright N \mid M \xrightarrow{\tau} \Omega \triangleright (\mathsf{new}\,\Phi)\, N' \mid M'\end{array}}$$

$$\text{(LTS-OPEN)} \quad \frac{\Omega \triangleright M \xrightarrow{(\Phi)a!V} \Omega' \triangleright M'}{\Omega \triangleright (\mathsf{new}\, b : \mathsf{E})\, M \xrightarrow{(b:\mathsf{E};\Phi)a!V} \Omega' \triangleright M'} \quad \begin{array}{l} b \neq a \\ b \in \mathsf{fn}(V) \cup \mathsf{fn}(\Phi) \end{array}$$

$$\text{(LTS-WEAK)} \quad \frac{\Omega; \Omega_e \triangleright M \xrightarrow{(\Phi)a?V} \Omega' \triangleright M'}{\Omega \triangleright M \xrightarrow{(\Omega_e;\Phi)a?V} \Omega' \triangleright M'} \quad \begin{array}{l} \mathsf{dom}(\Omega_e) \cap (\{a\} \cup \mathsf{fn}(M)) = \emptyset \\ \Omega_e \text{ is a loyal extension of } \Omega \end{array}$$

this equivalence: a bisimilarity. The idea of the bisimilarity is to provide an alternative but equivalent definition of the semantics using a *Labelled Transition System* (LTS) where the labels represent the possible interactions between the system and its environment. Then two systems can be distinguished if, after some preliminary interactions, one can perform a transition the other cannot.

The way the LTS is built is completely standard (see [1]): we associate the label τ to every internal reduction a system can perform, to indicate that the environment is not involved. The rule for migration (LTS-GOTO) is an example of this. Note that, since the interactions we are characterising are between some system M and an observer knowing Ω, we define transitions of *configurations* of the form $\Omega \triangleright M$. Also note that the knowledge of the observer is left untouched in a τ transition since it is not interacting with the system. In this extended abstract, we present in Figure 4 only the most significant rules, namely the rules where a message is exchanged with the environment and the rule for communication. The omitted rules are:

– the two natural contextual rules (for contexts of the forms $(\mathsf{new}\, a : \mathsf{E})[\cdot]$ and $[\cdot] \mid M$);
– and some τ transitions that can be directly derived from the reduction semantics, the way (LTS-GOTO) is obtained from (R-GOTO).

Let us explain (LTS-W). The conditions of this rule are similar to the ones for barbs. Indeed an observer knowing Ω will be able to interact with a system outputting a message V on a channel a in a location l only when l is reachable ($l \in \mathcal{R}_\Omega$) and when the observer can input on that channel ($\Omega \vdash_l a : \mathrm{R}\langle \mathsf{T} \rangle$).

The knowledge of the observer will consequently be enriched by the message: Ω becomes $\Omega,\langle V : \mathsf{T}\rangle$ along that transition. In this expression, the type T indicates all the rights the observer learns, calculated using the *meet* of the types associated with the channel. Suppose for instance that the meet of all the types associated with a in Ω is $\textsc{rw}\langle \mathsf{T}_1, \mathsf{T}_2\rangle@l$; then T_1 sums up all the rights that can be obtained by inputting on a. We denote that type T_1 as $\Omega^r(a)$ in (LTS-W).

With those transitions, we would like to define an equivalence $\mathcal{R}$ as a standard bisimulation: when $\Omega \vDash M \mathrel{\mathcal{R}} N$ and $\Omega \rhd M \xrightarrow{\mu} \Omega' \rhd M'$ then there must exist some N' such that $\Omega \rhd N \xrightarrow{\tau}{}^* \xrightarrow{\hat\mu} \xrightarrow{\tau}{}^* \Omega' \rhd N'$ and $\Omega' \vDash M' \mathrel{\mathcal{R}} N'$. But this definition cannot be used right away in our case, because of dependent types. Let us consider a case where the discrepancy appears. Suppose some channel c in l on which a passport can be transmitted (so c is of type $\textsc{rw}\langle \sum x, y : \vec{\textsc{loc}}.\ x \mapsto y\rangle@l$) and consider the following two systems:

$$(\mathsf{new}\, k' : \textsc{loc})\ (\mathsf{new}\, p : k, k' \mapsto l)\ l[\![c\,!\,\langle (k,l),(p)\rangle\ d\,!\,\langle k'\rangle]\!] \qquad (1)$$

$$(\mathsf{new}\, k' : \textsc{loc})\quad (\mathsf{new}\, p : k \mapsto l)\quad l[\![c\,!\,\langle (k,l),(p)\rangle\ d\,!\,\langle k'\rangle]\!] \qquad (2)$$

The only difference is the fact that the passport p can be used also from the new location k' in the first system. Since the observer receives p at the type $k \mapsto l$ in both cases, it should not be able to make the difference. But they can perform the following transitions with *distinct labels* (for simplicity, we ignore the type annotations in the labels):

$$\begin{array}{rrl}
\Omega \rhd (1) & \xrightarrow{(k',p)c!((k,l),(p))} & \Omega, p : k \mapsto l \rhd l[\![d\,!\,\langle k'\rangle]\!] \\
& \xrightarrow{d!k'} & \Omega, p : k \mapsto l, k' : \textsc{loc} \rhd l[\![\mathsf{stop}]\!] \\
\Omega \rhd (2) & \xrightarrow{(p)c!((k,l),(p))} & \Omega, p : k \mapsto l \rhd (\mathsf{new}\, k' : \textsc{loc})\, l[\![d\,!\,\langle k'\rangle]\!] \\
& \xrightarrow{(k')d!k'} & \Omega, p : k \mapsto l, k' : \textsc{loc} \rhd l[\![\mathsf{stop}]\!]
\end{array}$$

namely not opening the scope of k' in the same transition. To avoid this problem, we annotate configurations with a set of names whose scopes have been opened because of type dependencies, not because they were revealed. The labels are modified accordingly to mention only the names that are actually revealed.

Definition 5 (Actions). *The* annotated configuration $\Omega \rhd_{\tilde a} M$ *can perform the* action μ *and become* $\Omega' \rhd_{\tilde a'} M'$ *when:*

- *if μ is τ or $(\Phi)a?V$: the transition $\Omega \rhd M \xrightarrow{\mu} \Omega' \rhd M'$ is provable in the LTS and $\tilde a = \tilde a'$;*
- *if μ is $(\tilde b)a!V$: the transition $\Omega \rhd M \xrightarrow{(\Phi)a!V} \Omega' \rhd M'$ is provable in the LTS, $\tilde b = \mathsf{fn}(V) \cap (\mathsf{dom}(\Phi) \cup \tilde a)$ and $\tilde a' = (\mathsf{dom}(\Phi) \cup \tilde a) \setminus \mathsf{fn}(V)$.*

So, using an empty set as annotation, the first transition of the first system becomes:

$$\begin{array}{l}
\Omega \rhd_{\emptyset} (\mathsf{new}\, k' : \textsc{loc})\, (\mathsf{new}\, p : k, k' \mapsto l)\, l[\![c\,!\,\langle (k,l),(p)\rangle\ d\,!\,\langle k'\rangle]\!] \\
\qquad\qquad\qquad \xrightarrow{(p)c!((k,l),(p))} \Omega, p : k \mapsto l \rhd_{k'} l[\![d\,!\,\langle k'\rangle]\!]
\end{array}$$

Definition 6 (Loyal bisimilarity). *The* loyal bisimilarity*, written $\approx^{al}$, is the largest bisimulation defined in the standard way over* actions *of annotated configurations.*

The rest of this section is devoted to the proof that the two equivalences coincide under some conditions to justify that the bisimilarity has been introduced as a proof technique. The proof of that property is significantly more complex than its equivalent in the literature: the control of migrations hinders tracking the knowledge of the observer (apart from the passports, note that we must keep track of annotations because they hide some names from the observer). We will describe here only the most interesting aspects of the proof; more details are provided in [13], the proof is fully developed in [14].

The first step to bridge the gap between the loyal barbed congruence and the loyal bisimilarity is to account for annotations in configurations. This can be done by simply defining *annotated typed relations*, written

$$\Omega \vDash M \;_{\tilde{a}_M}\mathcal{S}_{\tilde{a}_N}\; N$$

and adapting the notion of contextuality to those relations. It is quite easy to see that the *annotated loyal barbed congruence* which ensues coincides with $\cong^l$ when its annotations are the empty sets of names.

The expected result then amounts to proving that the loyal bisimilarity and the annotated loyal barbed congruence coincide. The proof that the bisimilarity is included in the barbed congruence is mainly the proof of the fact that the bisimilarity is contextual. This is naturally done by checking all three items defining contextuality, the major property to check being:

Theorem 3 (Bisimilarity is closed on parallel contexts). *$\Omega \vDash M \;_{\tilde{a}_M}{\approx^{al}}_{\tilde{a}_N}\; N$, $l \in \mathcal{R}_\Omega$, $\Omega \vdash l[\![O]\!]$ and $\mathsf{fn}(O) \cap (\tilde{a}_M \cup \tilde{a}_N) = \emptyset$ imply $\Omega \vDash M \,|\, l[\![O]\!] \;_{\tilde{a}_M}{\approx^{al}}_{\tilde{a}_N}\; N \,|\, l[\![O]\!]$.*

Idea of proof. To get this result we simply build a relation and prove that it is a bisimulation which induces the fact that it is included in the biggest bisimulation, $\approx^{al}$. Because that relation must be closed on reductions, we will consider a relation $\mathcal{S}$ in which systems have a very general form:

$$\Omega \vDash (\mathsf{new}\,\Phi_M)(M \,|\, \prod_i l_i[\![O_i]\!]) \;_{\tilde{a}_M}\mathcal{S}_{\tilde{a}_N}\; (\mathsf{new}\,\Phi_N)(N \,|\, \prod_i l_i[\![O_i]\!])$$

The main difficulty to tackle is the fact that, along reductions, the knowledge of the observer, initially completely located in Ω (because $l[\![O]\!]$ is well-typed in Ω), is split between Ω and $\prod_i l_i[\![O_i]\!]$. In particular, a part of the environments Φ and annotations $\tilde{a}$ should be included in the general knowledge of the observer since they might have been communicated to the processes O_i. A precise account of this knowledge must be kept to preserve the full-strength of the initial hypothesis of bisimilarity between M and N. In particular, M and N must be bisimilar for an observer having access to all the locations l_i since it has some processes O_i running there. □

Let us now consider the converse, namely the fact that the congruence is included in the bisimilarity. The guiding idea of the definition of the actions was to identify all the possible interactions between a system and its observer. So the proof of that inclusion can be based on the definition of contexts that characterise a given action of the system. Those contexts use the fact that we can put any environment Γ in a normal form looking like:

$$\begin{aligned} & w_1 : \text{LOC}, \ldots, w_m : \text{LOC}, \\ & \qquad u_1 : \tilde{w}_{i_1} \mapsto w_{i_1}, \ldots, u_n : \tilde{w}_{i_n} \mapsto w_{i_n}, \\ & \qquad\qquad v_1 : \mathsf{C}_1@w_{j_1}, \ldots, v_o : \mathsf{C}_o@w_{j_o} \end{aligned}$$

where

- the w_k are all distinct;
- $u_k = u_{k'}$ only if $k = k'$ or if $w_{i_k} \neq w_{i_{k'}}$;
- $v_k = v_{k'}$ only if $k = k'$ or if $w_{j_k} \neq w_{j_{k'}}$.

So this normal form has the following structure: all the locations are defined first because types can depend only on location identifiers so that all the locations can be listed first; and every identifier is attributed exactly one type per location identifier to which it is attached[2]. The existence of such a normal form follows from the property of partial meets (Theorem 1) which ensures that all the types associated with a given identifier sum up to their meet.

This normal form of environments is relevant for the contexts that characterise the actions of a system because they provide a way to encode every environment into a value of the calculus.

Definition 7 (Reification of environments). *To an environment Γ of the following (normal) form*

$$\begin{aligned} & w_1 : \text{LOC}, \ldots, w_m : \text{LOC}, \\ & \qquad u_1 : \tilde{w}_{i_1} \mapsto w_{i_1}, \ldots, u_n : \tilde{w}_{i_n} \mapsto w_{i_n}, \\ & \qquad\qquad v_1 : \mathsf{C}_1@w_{j_1}, \ldots, v_o : \mathsf{C}_o@w_{j_o} \end{aligned}$$

we associate the value V_Γ and the type T_Γ such that:

$$\begin{aligned} V_\Gamma &= ((w_1, \ldots, w_m), (u_1, \ldots, u_n, v_1, \ldots, v_o)) \\ \mathsf{T}_\Gamma &= \textstyle\sum x_1, \ldots, x_m : \widetilde{\text{LOC}}.\ \tilde{x}^\star_{i_1} \mapsto x_{i_1}, \ldots, \tilde{x}^\star_{i_n} \mapsto x_{i_n}, \mathsf{C}_1@x_{j_1}, \ldots, \mathsf{C}_o@x_{j_o} \end{aligned}$$

Proposition 1 (Soundness of the reification). *For any well-formed environment Γ and any location w defined in Γ, $\Gamma \vdash_w V_\Gamma : \mathsf{T}_\Gamma$.*

Thanks to this reification of environments, we can proceed as usual, namely we can define some system $\mathfrak{C}^\Omega_\mathcal{N}((\tilde{b})c!U)$ so that $M \mid \mathfrak{C}^\Omega_\mathcal{N}((\tilde{b})c!U)$ will be sending the value $V_{\Omega,\langle U:\Omega^r(c)\rangle}$ on some specific channel ω if and only if the system M has actually sent the message U over the channel c to $\mathfrak{C}^\Omega_\mathcal{N}((\tilde{b})c!U)$. So such a context would be of the form $l[\![O]\!]$ where l is the location of the channel c in which the action takes place. Note that the observer can launch some process in l since the action $(\tilde{b})c!U$ is visible to the observer Ω: by rule (LTS-W) this implies that l is in $\mathcal{R}_\Omega$. Then O performs the following steps.

[2] When typechecking processes, a given channel or passport can be attached to more than one location variable.

1. It waits for a message on the channel c and, in parallel, exhibit a barb on some special channel δ.
2. It checks that the received value matches the expected U: this relies on the possibility to test the equality and inequality of names; in particular, to check that the names in $\tilde{b}$ are indeed fresh, the context is parameterised with a finite set of existing names $\mathcal{N}$ which contains all the names that are known to the observer. This test matches exactly the definition of the set $\tilde{b}$ in output actions: this set contains only the names which were hidden within the system or the annotation and which are revealed to the observer.
3. It finally cancels the barb on δ and outputs the value $V_{\Omega,\langle U:\Omega^r(c)\rangle}$ on the channel ω.

The channel δ used in the context serves only one purpose: to check that the step 2 has actually been performed: since the detected barbs always allow some preliminary τ transitions, the barb on ω is visible since the very beginning as soon as the system *can* perform the action.

By a very similar technique, it is possible to form contexts that characterise an input action, so that the following theorem can be proved:

Theorem 4. *The loyal annotated barbed congruence is included in the loyal bisimilarity.*

Idea of proof. We simply prove that $\cong^p$, the biggest annotated relation verifying the conditions of the loyal annotated barbed congruence apart from closure over $(\mathsf{new}\, a : \mathsf{E})[\cdot]$ contexts, is a bisimulation. For this consider $\Omega \vDash M\ {}_{\tilde{a}_M}{\cong^p}_{\tilde{a}_N}\ N$.

When the configuration $\Omega \rhd_{\tilde{a}_M} M$ performs a τ action to $\Omega \rhd_{\tilde{a}_M} M'$, the closure of $\cong^p$ on reductions gives a N' such that $\Omega \vDash M'\ {}_{\tilde{a}_M}{\cong^p}_{\tilde{a}_N}\ N'$.

For the action $\Omega \rhd_{\tilde{a}_M} M \xrightarrow{\alpha} \Omega' \rhd_{\tilde{a}'_M} M'$, we know that $M \,|\, \mathfrak{C}^{\Omega}_{\mathcal{N}}(\alpha)$ can reduce into some system $(\mathsf{new}\, \Phi_M)\, M' \,|\, \lambda[\![\omega\,!\,\langle V_{\Omega'}\rangle]\!]$. By contextuality and closure on reductions, $N \,|\, \mathfrak{C}^{\Omega}_{\mathcal{N}}(\alpha)$ should reach an equivalent state, with a barb on ω and no barb on δ. By definition of the context $\mathfrak{C}^{\Omega}_{\mathcal{N}}(\alpha)$, that equivalent state must be of the form $(\mathsf{new}\, \Phi_N)\, N' \,|\, \lambda[\![\omega\,!\,\langle V_{\Omega'}\rangle]\!]$ with $\Omega \rhd_{\tilde{a}_N} N \overset{\alpha}{\Longrightarrow} \Omega' \rhd_{\tilde{a}'_N} N'$.

A fairly standard *scope extrusion lemma* (see for instance [1]) bridges the last gap by concluding $\Omega' \vDash M'\ {}_{\tilde{a}'_M}{\cong^p}_{\tilde{a}'_N}\ N'$ from

$$\lambda, \omega, \pi \vDash (\mathsf{new}\, \Phi_M)\, M' \,|\, \lambda[\![\omega\,!\,\langle V_{\Omega'}\rangle]\!]\ {}_{\tilde{a}'_M}{\cong^p}_{\tilde{a}'_N}\ (\mathsf{new}\, \Phi_N)\, N' \,|\, \lambda[\![\omega\,!\,\langle V_{\Omega'}\rangle]\!]$$

where: π is a universal passport to λ, $\tilde{a}'_M$ is $(\tilde{a}_M \cup \mathsf{dom}(\Phi_M)) \setminus \mathsf{dom}(\Omega)$ and a similar formula for $\tilde{a}'_N$. $\square$

The results stated above directly entails the expected result:

Theorem 5 (Full abstraction of $\approx^{al}$ for $\cong^l$). *$\Omega \vDash M \cong^l N$ if and only if $\Omega \vDash M\ {}_{\emptyset}{\approx^{al}}_{\emptyset}\ N$*

5 Conclusion and Perspectives

This work presents a new approach to control the migrations of agents in the context of distributed computation, using simple passports that should correspond to the origin location of the migrating agent. We have developed the full theory of this idea, with a loyal barbed congruence that takes those passports into account to distinguish between systems. We have also provided a complete proof technique for this equivalence as a bisimilarity.

This work provides a solid ground on which to investigate subtler notions of security like the ones presented in [9] and [18]. We already started to study more complex passports in which resources that can be accessed after the migration depend on the passport actually used: when a new passport is generated, its type also embed all the rights to be granted to incoming processes.

It would also be interesting to refine passports to stricter notions of trust, where other locations are prevented from relaying processes for instance.

Acknowledgement. The author would like to thank Matthew Hennessy for numerous helpful discussions and comments.

References

1. Hennessy, M.: A Distributed Pi-calculus. Cambridge University Press, Cambridge (2007)
2. Levi, F., Sangiorgi, D.: Controlling interference in ambients. In: 27th Annual Symposium on Principles of Programming Languages (POPL), Boston, MA, pp. 352–364. ACM Press, New York (2000)
3. Merro, M., Hennessy, M.: A bisimulation-based semantic theory of Safe Ambients. ACM Transactions on Programming Languages and Systems 28(2), 290–330 (2006)
4. Bugliesi, M., Crafa, S., Merro, M., Sassone, V.: Communication interference in mobile boxed ambients. In: Agrawal, M., Seth, A.K. (eds.) FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science. LNCS, vol. 2556, pp. 71–84. Springer, Heidelberg (2002)
5. Castagna, G., Nardelli, F.Z.: The Seal calculus revisited: Contextual equivalence and bisimilarity. In: Agrawal, M., Seth, A.K. (eds.) FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science. LNCS, vol. 2556, pp. 85–96. Springer, Heidelberg (2002)
6. Schmitt, A., Stefani, J.-B.: The Kell calculus: A family of higher-order distributed process calculi. In: Priami, C., Quaglia, P. (eds.) GC 2004. LNCS, vol. 3267, pp. 146–178. Springer, Heidelberg (2005)
7. Hennessy, M., Merro, M., Rathke, J.: Towards a behavioural theory of access and mobility control in distributed systems. Theoretical Computer Science 322, 615–669 (2003)
8. Cardelli, L., Ghelli, G., Gordon, A.D.: Ambient groups and mobility types. In: Watanabe, O., Hagiya, M., Ito, T., van Leeuwen, J., Mosses, P.D. (eds.) TCS 2000. LNCS, vol. 1872, pp. 333–347. Springer, Heidelberg (2000)
9. Hennessy, M., Rathke, J., Yoshida, N.: SafeDpi: a language for controlling mobile code. Acta Informatica 42(4-5), 227–290 (2005)
10. Yoshida, N.: Channel dependent types for higher-order mobile processes (September 2004)
11. Martins, F., Vasconcelos, V.T.: History-based access control for distributed processes. In: De Nicola, R., Sangiorgi, D. (eds.) TGC 2005. LNCS, vol. 3705, pp. 98–115. Springer, Heidelberg (2005)

12. Boreale, M., Sangiorgi, D.: Bisimulation in name-passing calculi without matching. In: Thirteenth Annual Symposium on Logic in Computer Science (LICS) (Indiana), IEEE Computer Society Press, Los Alamitos (1998)
13. Hym, S.: Mobility control via passports (preprint 2007), Available on https://hal.archives-ouvertes.fr/hal-00140527 http://www.pps.jussieu.fr/~hym/r/
14. Hym, S.: Typage et contrôle de la mobilité. PhD thesis, Université Paris Diderot – Paris 7 (December 2006)
15. Hennessy, M., Riely, J.: Resource access control in systems of mobile agents. Information and Computation 173, 82–120 (2002)
16. Hym, S., Hennessy, M.: Adding recursion to Dpi. Theoretical Computer Science 373(3), 182–212 (2007)
17. Milner, R., Sangiorgi, D.: Barbed bisimulation. In: Kuich, W. (ed.) Automata, Languages and Programming. LNCS, vol. 623, pp. 685–695. Springer, Heidelberg (1992)
18. Crary, K., Harper, R., Pfenning, F., Pierce, B.C., Weirich, S., Zdancewic, S.: Manifest security for distributed information. White paper (March 2006)

A Reduction Semantics

Fig. 5. Reduction semantics

$$\text{(R-GOTO)}\quad l[\![\mathsf{goto}_p\, k.\ P]\!] \longrightarrow k[\![P]\!]$$

$$\text{(R-NEWLOC)}\quad l[\![\mathsf{newloc}\, k, (\vec{c}), (\vec{p}), (\vec{q}) : \textstyle\sum x : \text{LOC}.\ \mathsf{T}\ \mathsf{with}\ P_k\ \mathsf{in}\ P]\!]$$
$$\longrightarrow (\mathsf{new}\big\langle k, ((\vec{c}), (\vec{p}), (\vec{q})) : \mathsf{T}\{^l\!/\!_x\}\big\rangle)(k[\![P_k]\!] \mid l[\![P]\!])$$

$$\text{(R-NEWPASS)}\quad l[\![\mathsf{newpass}\, p\ \mathsf{from}\ \tilde{k}^\star\ \mathsf{in}\ P]\!] \longrightarrow (\mathsf{new}\, p : \tilde{k}^\star \mapsto l)\, l[\![P]\!]$$

$$\text{(R-COMM)}\quad l[\![a\,!\,\langle V\rangle\, P_1]\!] \mid l[\![a\,?\,(X : \mathsf{T})\, P_2]\!] \longrightarrow l[\![P_1]\!] \mid l[\![P_2\{^V\!/x\}]\!]$$

$$\text{(R-IF-V)}\quad l[\![\mathsf{if}\ a = a\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2]\!] \longrightarrow l[\![P_1]\!]$$

$$\text{(R-IF-F)}\quad l[\![\mathsf{if}\ a_1 = a_2\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2]\!] \longrightarrow l[\![P_2]\!] \quad \text{when } a_1 \neq a_2$$

$$\text{(R-NEWCHAN)}\quad l[\![\mathsf{newchan}\, c : \mathsf{C}\ \mathsf{in}\ P]\!] \longrightarrow (\mathsf{new}\, c : \mathsf{C}@l)\, l[\![P]\!]$$

$$\text{(R-SPLIT)}\quad l[\![P_1 \mid P_2]\!] \longrightarrow l[\![P_1]\!] \mid l[\![P_2]\!]$$

$$\text{(R-REP)}\quad l[\![*P]\!] \longrightarrow l[\![P]\!] \mid l[\![*P]\!]$$

$$\text{(R-C-PAR)}\quad \frac{M_1 \longrightarrow M_1'}{M_1 \mid M_2 \longrightarrow M_1' \mid M_2} \qquad \text{(R-C-NEW)}\quad \frac{M_1 \longrightarrow M_1'}{(\mathsf{new}\, a : \mathsf{E})\, M_1 \longrightarrow (\mathsf{new}\, a : \mathsf{E})\, M_1'}$$

$$\text{(R-STRUCT)}\quad \frac{M_1 \equiv M_2 \longrightarrow M_2' \equiv M_1'}{M_1 \longrightarrow M_1'}$$

Coalgebraic Models for Reactive Systems[⋆]

Filippo Bonchi and Ugo Montanari

Dipartimento di Informatica, Università di Pisa

Abstract. Reactive Systems *à la* Leifer and Milner allow to derive from a reaction semantics definition an LTS equipped with a bisimilarity relation which is a congruence. This theory has been extended by the authors (together with Barbara König) in order to handle saturated bisimilarity, a coarser equivalence that is more adequate for some interesting formalisms, such as logic programming and open pi-calculus. In this paper we recast the theory of Reactive Systems inside Universal Coalgebra. This construction is particularly useful for saturated bisimilarity, which can be seen as final semantics of Normalized Coalgebras. These are structured coalgebras (not bialgebras) where the sets of transitions are minimized rather than maximized as in saturated LTS, still yielding the same semantics. We give evidence the effectiveness of our approach minimizing an Open Petri net in a category of Normalized Coalgebras.

1 Introduction

The operational semantics of process calculi has traditionally been specified by labelled transition systems (LTSs), and the abstract semantics by bisimilarity relations defined on them. Bisimilarities often turn out to be congruences with respect to the operations of the languages, a property which expresses the compositionality of the abstract semantics. A simpler approach, inspired by classical formalisms like λ-calculus, Petri nets, term and graph rewriting - pioneered by the Chemical Abstract Machine [3] and especially convenient for nominal calculi - defines operational semantics by means of *structural axioms* and *reaction rules*. Transitions caused by reaction rules, however, are not labeled, since they represent evolutions of the system without interactions with the external world. Thus reaction semantics is neither abstract nor compositional.

To enhance the expressiveness of reaction semantics, Leifer and Milner proposed in [12] the *theory of reactive systems*: a systematic method for deriving a labeled transition system from reaction rules. The main idea is the following: a process p can do a move with label $C[-]$ and become p' iff there is a reaction rule transforming $C[p]$ in p'. This LTS is called *Context Transitions System* (*CTS*) and the bisimilarity over it ($\sim_{SAT}$, called *saturated*) is always a congruence. However, such an LTS is usually very large, typically infinite branching and overloaded with redundant transitions, since often contexts $C[-]$ contain components which are irrelevant for the transition.

For this reason, Leifer and Milner introduced the notions of relative pushout (RPO) and idem relative pushout (IPO) for specifying a/the minimal context that allows the

⋆ Research partially supported by the IST 2004-16004 SENSORIA, and the MIUR PRIN 2005015824 ART.

state to react with a rule. This construction leads to the *IPO transition system* (*ITS*), that uses only contexts generated by IPOs, and not all contexts, as labels, and thus is smaller than *CTS*. Bisimilarity on this LTS ($\sim_{IPO}$) is a congruence under restrictive conditions.

In [4], the authors proved that for some interesting formalisms, such as logic programming and open π-calculus, $\sim_{IPO}$ is at some extent inadequate, since it is strictly finer than standard abstract semantics, while $\sim_{SAT}$ exactly characterizes it.

Universal Coalgebra [15] provides a categorical framework where abstract semantics of interactive computing systems are described as morphisms to their minimal representatives. More precisely, given an endofunctor $\mathbf{F}$ on a category $\mathbf{C}$, a coalgebra is an arrow $f\colon\ X \to \mathbf{F}(X)$ of $\mathbf{C}$ and a coalgebra morphism from f to f' is an arrow $h\colon\ X \to X'$ of $\mathbf{C}$ with $h\ ; f' = f\ ; \mathbf{F}(h)$. Under certain conditions on $\mathbf{C}$ and $\mathbf{F}$, a category of coalgebras admits a final object. Ordinary labeled transition systems (with finite or countable branching) can be represented as coalgebras with final object for a suitable functor on $\mathbf{Set}$. Then, in order to prove that two states are equivalent, we have to check if they are identified by the final morphism, and the image of the given coalgebra through the latter is the minimal representative, which in the finite case can be usually computed via the list partitioning algorithm by Kanellakis and Smolka [9].

However, this representation of interactive systems forgets about the algebraic structure, which is usually very relevant in practical cases, since compositionality is the key to master complexity. In particular, the property that bisimilarity respects the operations, i.e. that it is a congruence, which is essential for making abstract semantics compositional, is not reflected in the structure of the model.

In [19], *bialgebras* are introduced as a model with both algebraic and coalgebraic structure, while a related approach based on *structured coalgebras* is presented in [7]. In the latter work, the endofunctor determining the coalgebraic structure is lifted from $\mathbf{Set}$ to the category of Σ-algebras, for some algebraic signature Σ. Morphisms between coalgebras in this category are both Σ-homomorphisms and coalgebra morphisms: as a consequence the unique morphism to the final coalgebra always induces a bisimilarity that is a congruence.

In this paper we provide a structured coalgebraic construction for both *ITS* and *CTS*. This is interesting for at least two reasons. On the one hand it assures the existence of final semantics and minimal representatives for reactive systems, both for the Leifer and Milner's IPO semantics $\sim_{IPO}$ and for our saturated semantics $\sim_{SAT}$. On the other hand it is an alternative compositionality proof of them.

For practical applications a key issue is how efficiently our saturated semantics can be computed, which, according to its definition, is based on the large and redundant *CTS*. In the previous paper [4], an unconventional notion of bisimulation, called *semisaturated*, is presented for this purpose. It allows Alice, the first player of the bisimulation game, to choose a transition in *ITS*, while Bob, the second player, chooses in *CTS*. Semisaturated bisimilarity is the same as saturated bisimilarity, but the size of the game is much smaller. Unfortunately, this approach cannot be extended to coalgebraic theory, since in the latter case there is only one transition system for both players.

A further contribution of the paper is the construction of yet another LTS which has fewer transitions than *ITS*, but supports $\sim_{SAT}$. In reactive systems all non-IPO transitions, i.e. the transitions labeled with a context that is not strictly necessary to perform a transition with a given rule, are considered redundant and omitted. Here we introduce a stronger notion of redundancy. Indeed we consider redundant the transitions $p \overset{C[-]}{\rightarrow} p'$ such that $p \overset{D[-]}{\rightarrow} p''$, and $D[-], p''$ are smaller, i.e., there exists a context $E[-]$ such that $E[D[-]] = C[-]$ and $E[p''] \sim p'$. Thus our notion of redundancy is based on bisimilarity and it is independent from the rule that allows the reaction, while the IPO construction is based on syntactic equivalence and it is relative to a particular rule.

Our construction is based on *Normalized Coalgebras*. These are structured coalgebras without redundant transitions which form a category with final object, where the unique morphism induces a notion of bisimilarity completely abstract from redundant transitions. We prove that the category of Normalized Coalgebras is isomorphic to the category of saturated coalgebras (the coalgebras containing all the redundant transitions), where the large context transition system *CTS* can be directly modelled. In doing this, we use the notions of *normalization* that junks away all the redundant transitions, and of *saturation* that adds all the redundant transitions. Both are natural transformations between the functors (defining the two categories of coalgebras) and one is the inverse of the other. As a corollary of the isomorphism theorem, $\sim_{SAT}$ can be characterized as bisimilarity in the category of Normalized Coalgebras. This proves that our notion of non-redundancy is more canonical than IPOs, since it exactly captures $\sim_{SAT}$.

Normalized Coalgebras provide an efficient way to compute $\sim_{SAT}$. Indeed we can forget about all the redundant transitions (obtaining a labeled transition system smaller than *ITS*) and then we can compute the final morphism in the category of normalized coalgebras, through the canonical minimization algorithm. Normalized Coalgebras turn out to be theoretically interesting for one additional reason. Those are, to our knowledge, the first interesting example of structured coalgebras that are not bialgebras.

Synopsis. In Sec. 2 and 3, we introduce the theory of reactive systems and (structured) coalgebras. Then in Sec. 4 and 5 we provide a structured coalgebraic construction for *CTS* and for *ITS*. In Sec. 6 we introduce Normalized Coalgebras, a minimization algorithm for these and we apply it to a concrete example.

2 The Theory of Reactive Systems

Here we summarize the theory of reactive systems proposed in [12] to derive labelled transition systems and bisimulation congruences from a given reaction semantics. The theory is centred on the concepts of *term*, *context* and *reaction rules*: contexts are arrows of a category, terms are arrows having as domain 0 (a special object that denotes no holes), and reaction rules are pairs of terms.

Definition 1 (Reactive System). *A* reactive system $\mathcal{R}$ *consists of:*

1. *a category* $\mathbf{C}$
2. *a distinguished object* $0 \in |\mathbf{C}|$
3. *a composition-reflecting subcategory* $\mathbf{D}$ *of* reactive contexts
4. *a set of pairs* $\mathbb{R} \subseteq \bigcup_{m \in |\mathbf{C}|} \mathbf{C}[0, m] \times \mathbf{C}[0, m]$ *of* reaction rules.

The reactive contexts are those in which a reaction can occur. By composition-reflecting we mean that $d; d' \in \mathbf{D}$ implies $d, d' \in \mathbf{D}$.

From reaction rules one generates the reaction relation by closing them under all reactive contexts. Formally, the *reaction relation* is defined by taking $p \rightsquigarrow q$ if there is $\langle l, r\rangle \in \mathbb{R}$ and $d \in \mathbf{D}$ such that $p = l; d$ and $q = r; d$.

Thus the behaviour of a reactive system is expressed as an unlabelled transition system. On the other hand many behavioural equivalences are only defined for LTSs. In order to obtain an LTS, we can plug a term p into some context c and observe if a reaction occurs. In this case we have that $p \xrightarrow{c}$. Categorically speaking this means that $p; c$ matches $l; d$ for some rule $\langle l, r\rangle \in \mathbb{R}$ and some reactive context d. This situation is depicted by diagram (i) in Fig. 1: a commuting diagram like this is said a *redex square*.

Definition 2. *The* context transition system *(CTS for short) is defined as follows:*

– *states: arrows $p : 0 \to m$ in* **C***, for arbitrary m;*
– *transitions: $p \xrightarrow{c}_C q$ iff $p; c \rightsquigarrow q$.*

Bisimilarity on this LTS is called *saturated* (denoted by $\sim_{SAT}$), and it is always a congruence (i.e., preserved under all contexts). However this labelled transition system is often infinite-branching since all contexts that allow reactions may occur as labels. Another problem of *CTS* is that it has redundant transitions. For example, consider the term $a.0$ of CCS. The observer can put this term into the context $\overline{a}.0 \mid -$ and observe a reaction. This corresponds to the transition $a.0 \xrightarrow{\overline{a}.0|-}_C 0|0$. However we also have $a.0 \xrightarrow{p|\overline{a}.0|-}_C p \mid 0 \mid 0$ as a transition, yet p does not contribute to the reaction. Hence we need a notion of "minimal context that allows a reaction". Leifer and Milner define idem pushouts (IPOs) in order to capture this notion.

Definition 3 (RPO/IPO). *Let the diagrams (ii)-(v) in Fig. 1 be in some category* **C**. *Let (ii) be commuting. Any tuple $\langle x, e, f, g\rangle$ which makes (iii) commute is called a* candidate *for (ii). A* relative pushout (RPO) *is the smallest such candidate. More formally, it satisfies the universal property that given any other candidate $\langle y, e', f', g'\rangle$, there exists a unique mediating morphism $h : x \to y$ such that (iv) and (v) commute.*

Diagram (ii) of Fig. 1 is called idem pushout (IPO) *if $\langle o, c, d, id_o\rangle$ is an RPO.*

$$\begin{tikzcd}
& n & \\
k \arrow[ur, "c"] & & j \arrow[ul, "d"'] \\
& 0 \arrow[ul, "p"] \arrow[ur, "l"'] &
\end{tikzcd}$$
(i)

$$\begin{tikzcd}
& o & \\
m \arrow[ur, "c"] & & n \arrow[ul, "d"'] \\
& k \arrow[ul, "a"] \arrow[ur, "b"'] &
\end{tikzcd}$$
(ii)

$$\begin{tikzcd}
& o & \\
m \arrow[ur, "c"] \arrow[r, "e"'] & x \arrow[u, "g"] & n \arrow[ul, "d"'] \arrow[l, "f"] \\
& k \arrow[ul, "a"] \arrow[ur, "b"'] &
\end{tikzcd}$$
(iii)

$$\begin{tikzcd}
& y & \\
m \arrow[ur, "e'"] \arrow[r, "e"'] & x \arrow[u, "h"] & n \arrow[ul, "f'"'] \arrow[l, "f"]
\end{tikzcd}$$
(iv)

$$\begin{tikzcd}
& o \\
y \arrow[ur, "g'"] & x \arrow[u, "g"'] \arrow[l, "h"]
\end{tikzcd}$$
(v)

$$\begin{tikzcd}
& n & y \arrow[l, "e"'] & \\
k \arrow[ur, "c"] \arrow[urr, "c'"'] & & j \arrow[ul, "d"] & x \arrow[ul, "d'"'] \\
& 0 \arrow[ul, "p"] \arrow[ur, "l"] \arrow[urr, "l'"'] & &
\end{tikzcd}$$
(vi)

Fig. 1. Redex Square and RPO

We say that a reactive system *has (redex) RPOs* if, in the underlying category, for each (redex) square there exists an RPO, while it *has (redex) IPOs*, if every (redex) square has at least one IPO as candidate. A deeper discussion about the relationship between the two concepts can be found in [4].

Definition 4. *The* IPO transition system *(ITS for short) is defined as follows:*

- *states: $p : 0 \to m$ in $\mathbf{C}$, for arbitrary m;*
- *transitions: $p \xrightarrow{c}_I r; d$ iff $d \in \mathbf{D}$, $\langle l, r\rangle \in \mathbb{R}$ and the diagram (i) in Fig. 1 is an IPO.*

In other words, if inserting p into the context c matches $l; d$, and c is the "smallest" such context (according to the IPO condition), then p transforms to $r; d$ with label c, where r is the reduct of l. Bisimilarity on *ITS* is referred to as *standard bisimilarity* (denoted by $\sim_{IPO}$), and [12] proves the following.

Proposition 1. *In reactive systems with redex RPOs, $\sim_{IPO}$ is a congruence.*

In [4], the authors, together with Barbara König, show that $\sim_{IPO}$ is usually finer than $\sim_{SAT}$, and the latter is more appropriate than the former in some important cases: in Logic Programming and Open π-calculus, saturated semantics capture the canonical abstract semantics (i.e., logic equivalence and open bisimilarity), while standard semantics result too fine. Since *CTS* is full of redundancy (and usually infinite branching), the authors introduce semi saturated bisimulation to efficiently characterize $\sim_{SAT}$.

Definition 5 (Semi-Saturated Bisimulation). *A symmetric relation R is a semi-saturated bisimulation iff whenever $p\,R\,q$,*
if $p \xrightarrow{c}_I p'$ then $q \xrightarrow{d}_I q'$ and $\exists e \in \mathbf{D}$ such that $d; e = c$ and $p'\,R\,q'; e$.
The union of all Semi-Saturated bisimulations is Semi-Saturated bisimilarity *($\sim_{SS}$).*

This characterization is more efficient than considering all the possible contexts as labels. Nevertheless, as the following proposition states, it coincides with $\sim_{SAT}$.

Proposition 2. *In reactive systems with redex IPOs, $\sim_{SAT} = \sim_{SS}$.*

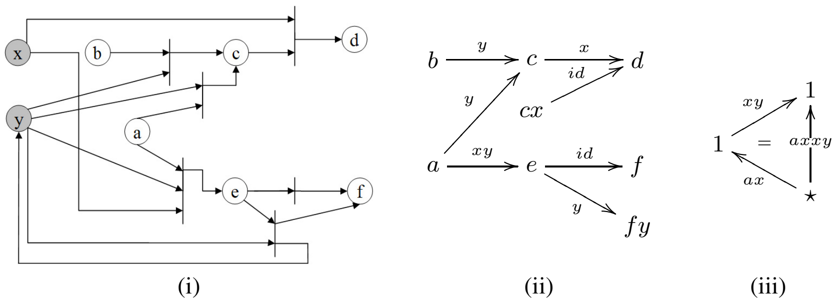


Fig. 2. (i) The Open Petri net $\mathcal{N}$. (ii) The *ITS* of a, b and cx. (iii) Arrows composition in $\mathbf{OPL}$.

Example 1. Open Petri nets [10,2] are P/T nets equipped with an *interface*, i.e., a set of *open places*, where nets can receive tokens from the environment[1]. Consider the Open net in Fig. 2(i). The interface of this set is the set of open places x and y depicted in gray. This net defines the reactive system $\mathcal{N} = \langle \mathbf{OPL}, \star, \mathbf{OPL}, \mathbb{T}\rangle$. Roughly the states of

[1] [13] encodes C/E nets into Bigraphs [14], while [16] P/T nets into Borrowed Contexts [8].

OPL (arrows from $\star$ to 1) are multisets on all the places, while contexts (arrows from 1 to 1) are multisets on open places. The composition of a state m_1 with a context m_2 is defined as the union of the multisets m_1 and m_2 as shown in Fig. 2(iii). Every transition of the net describes a reaction rule, where the left hand side is the precondition of the transition, and the right hand side is the postcondition.

Hereafterwe will use id for the empty multiset and aab for the multiset $\{a, a, b\}$. The *ITS* of a and b is depicted in Fig.2(ii). Consider the multisets e and cx. The former can interact both with the rule $\langle e, f\rangle$ generating the transition $e \xrightarrow{id}_I f$ and with the rule $\langle ey, fy\rangle$ generating the transition $e \xrightarrow{y}_I fy$. The latter can interact only with the rule $\langle cx, d\rangle$ generating the transition $cx \xrightarrow{id}_I d$. Thus $e \nsim_{IPO} cx$, but $e \sim_{SAT} cx$. Indeed the *CTS* move $e \xrightarrow{y}_C fy$ is matched by $cx \xrightarrow{y}_C dy$ and $fy \sim_{SAT} dy$ since both cannot move. Moreover $a \nsim_{IPO} b$ but $a \sim_{SS} b$ (and thus $a \sim_{SAT} b$). Indeed when a proposes $a \xrightarrow{xy}_I e$, b can answer with $b \xrightarrow{y}_I c$ and, as proved above, $e \sim_{SAT} cx$.

3 Coalgebras and Structured Coalgebras

In this section we introduce first the basic notions of the theory of coalgebras [15] and then structured coalgebras [7] in order to model reactive systems.

Definition 6 (coalgebras and cohomomorphisms). *Let* $\mathbf{F} : \mathbf{C} \to \mathbf{C}$ *be an endofunctor on a category* $\mathbf{C}$. *A* coalgebra for $\mathbf{F}$ *or (*$\mathbf{F}$-coalgebra*) is a pair* $\langle A, \alpha\rangle$ *where* A *is an object of* $\mathbf{C}$ *and* $\alpha : A \to \mathbf{F}(A)$ *is an arrow. An* $\mathbf{F}$-cohomomorphism $f : \langle A, \alpha\rangle \to \langle B, \beta\rangle$ *is an arrow* $f : A \to B$ *of* $\mathbf{C}$ *such that* $f; \beta = \alpha; \mathbf{F}(f)$.

$\mathbf{Coalg_F}$ *is the category of* $\mathbf{F}$*-coalgebras and* $\mathbf{F}$*-cohomomorphisms.*

Let $\mathbf{P_L} : \mathbf{Set} \to \mathbf{Set}$ be the functor defined as $X \mapsto \mathbf{P}(L \times X)$ where L is a fixed set of labels and $\mathbf{P}$ denotes the powerset functor. Then coalgebras for this functor are one-to-one with labeled transition systems over L [15]. Transition system morphisms are usually defined as functions between the carriers that *preserve* transitions, while $\mathbf{P_L}$-cohomomorphisms not only preserve, but also *reflect* transitions.

We can give a categorical characterization of bisimilarity if there exists a *final coalgebra*. Two elements of the carrier of a coalgebra are bisimilar iff they are mapped to the same element of the final coalgebra by the unique cohomomorphism. Indeed, in the final coalgebra all bisimilar states are identified, and thus, the image of a coalgebra through the unique morphism, is the minimal realization (w.r.t. bisimilarity) of the coalgebra. Computing the unique morphism just means to minimize the coalgebras, that it is usually possible in the finite case, using the following algorithm [1]:

1. Given a $\mathbf{F}$-coalgebra $\langle X, \alpha\rangle$, we initialize $!_0 : X \to 1$ as the morphism that maps all the elements of X into the one element set 1. This represents the trivial partitioning where all the elements are considered equivalent.
2. Then $!_{n+1}$ is defined as $\alpha; \mathbf{F}(!_n)$. This function defines a new finer partition on X. If the partition is equivalent to that of $!_n$, then this partition equates all and only the bisimilar states (i.e., coincides with !).
   If the set of states is finite, then the algorithm terminates.

$$\begin{tikzcd}
X \arrow[r, "!_n"] \arrow[d, "\alpha"'] \arrow[dr, "!_{n+1}"] & F^n(1) \\
F(X) \arrow[r, "F(!_n)"'] & F^{n+1}(1)
\end{tikzcd}$$

Unfortunately, due to cardinality reasons, the category of $\mathbf{P_L}$-coalgebras does not have a final object [15]. One satisfactory solution consists of replacing the powerset functor $\mathbf{P}$ by the *countable* powerset functor $\mathbf{P}_c$, which maps a set to the family of its countable subsets. Then defining the functor $\mathbf{P^c_L} : \mathbf{Set} \to \mathbf{Set}$ by $X \mapsto \mathbf{P}_c(L \times X)$ one has that coalgebras for this endofunctor are one-to-one with transition systems with *countable degree*. Unlike functor $\mathbf{P_L}$, functor $\mathbf{P^c_L}$ admits final coalgebras (Ex. 6.8 of [15]).

The coalgebraic representation using functor $\mathbf{P^c_L}$ is not completely satisfactory, because by definition the carrier of a coalgebra is just a set and therefore the intrinsic algebraic structure of states is lost. This calls for the introduction of *structured coalgebras*, i.e., coalgebras for an endofuctor on a category $\mathbf{Alg_\Gamma}$ of algebras for a specification Γ. Since cohomomorphisms in a category of structured coalgebras are also Γ-homomorphisms bisimilarity is a congruence w.r.t. the operations in Γ.

In [19], bialgebras are used as structures combining algebras and coalgebras. Bialgebras are richer than structured coalgebras, since they can be seen both as coalgebras on algebras and also as algebras on coalgebras. Categories of bialgebras over the functor $\mathbf{P^c_L}$ have a final object and bisimilarity abstracts from the algebraic structure.

In [6], it is proved that whenever the endofunctor on algebras is a lifting of $\mathbf{P^c_L}$, then structured coalgebras coincide with bialgebras.

Proposition 3. *Let Γ be an algebraic specification. Let $\mathbf{V^\Gamma} : \mathbf{Alg_\Gamma} \to \mathbf{Set}$ be the forgetful functor. If $\mathbf{F_\Gamma} : \mathbf{Alg_\Gamma} \to \mathbf{Alg_\Gamma}$ is a* lifting of $\mathbf{P^c_L}$ along $\mathbf{V^\Gamma}$ *(i.e., $\mathbf{F_\Gamma};\mathbf{V^\Gamma} = \mathbf{V^\Gamma};\mathbf{P^c_L}$), then $\mathbf{F_\Gamma}$-coalgebras are bialgebras and $\mathbf{Coalg_{F_\Gamma}}$ has a final object.*

4 Coalgebraic Models of *CTS*s

In this section we give a coalgebraic characterization of Context Transition Systems of reactive systems through the theory outlined in the previous section. We will first define the *CTS* as a coalgebra without algebraic structure and then we will lift it to a structured setting. This proves that bisimilarity on *CTS* (i.e., $\sim_{SAT}$) is a congruence, and moreover it provides a characterization of $\sim_{SAT}$ as final semantics.

Firstly we have to define the universe of observations. Since the labels of the *CTS* are arrows of the base category $\mathbf{C}$ (representing the contexts), we define the functor as parametric w.r.t. $\mathbf{C}$, and $||\mathbf{C}||$ (i.e. the class of all arrows of $\mathbf{C}$) is the universe of labels.

Definition 7. *Given a category $\mathbf{C}$, the functor $\mathbf{P_C} : \mathbf{Set}^{|\mathbf{C}|} \to \mathbf{Set}^{|\mathbf{C}|}$ is defined for each $|\mathbf{C}|$-indexed set S by $\mathbf{P_C}(S_n) = \mathbf{P}_c\left(\bigcup_{m\in|\mathbf{C}|} \mathbf{C}[n,m] \times S_m\right)$.*
The functor is defined analogously on arrows of $\mathbf{Set}^{|\mathbf{C}|}$.

Note that $\mathbf{P_C}$ is not an endofunctor on $\mathbf{Set}$, as it is the case for the standard $\mathbf{P_L}$ discussed in the previous section, but it is defined on $\mathbf{Set}^{|\mathbf{C}|}$, i.e., the category of sets indexed by objects of $\mathbf{C}$. The base category $\mathbf{C}$ induces $\overline{\mathbf{C}}$, an object of $\mathbf{Set}^{|\mathbf{C}|}$ where, for any sort n, the corresponding set is $\mathbf{C}[0,n]$. Here we have implicitly assumed that $\mathbf{C}$ is *locally small* (i.e., the hom-class between two objects is always a set and not a proper class), otherwise $\mathbf{C}[0,n]$ could be a proper class. Moreover, in the following definition, we require that $||\mathbf{C}||$ is a countable set, otherwise the possible transitions of an element could beuncountable and then not belong to $\mathbf{P_C}$. Note that this usually holds in those categories where arrows are syntactic contexts of a formalism.

Definition 8. *Given a reactive system $\mathcal{R} = \langle \mathbf{C}, 0, \mathbf{D}, \mathbb{R}\rangle$, the $\mathbf{P_C}$-coalgebra corresponding to its CTS is $\langle \overline{\mathbf{C}}, \alpha_{\mathcal{R}}\rangle$ where $\alpha_{\mathcal{R}}(p) = \{(c, r; d)$ s.t. diagram (i) in Fig. 1 commutes and $d \in \mathbf{D}$ and $\langle l, r\rangle \in \mathbb{R}\}$.*

It is immediate to see that the LTS defined above exactly coincides with the *CTS* (Def. 2). However this model does not take into account the algebraic structure of the states, i.e., of the possibility of contextualizing a term. In order to have a richer model we lift this construction to a structured setting where the base category is not anymore $\mathbf{Set}^{|\mathbf{C}|}$, but a category of algebras with contextualization operations. In the following we assume that the category $\mathbf{C}$ has *strict distinguished object*, i.e., that the only arrow with target 0 is id_0. This is needed to distinguish between elements and operations of algebras.

specification $\Gamma(\mathcal{R})$ =
  sorts
    n $\qquad\qquad$ $\forall n \in |\mathbf{C}|$ with $n \neq 0$
  operations
    $d : n \to m$ $\qquad$ $\forall d \in \mathbf{C}[n, m]$ with $n \neq 0$
  equations
    $id(x) = x$
    $e(d(x)) = c(x)$ $\qquad$ $\forall d; e = c$

This signature defines $\mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$ the category of $\Gamma(\mathcal{R})$-algebras. The base category $\mathbf{C}$ of a reactive system induces $\widehat{\mathbf{C}} \in |\mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}|$. In $\widehat{\mathbf{C}}$, for every sort m, the elements of this sort are the arrows of $\mathbf{C}[0, m]$. Every operation $c : m \to n$ is defined for every element p of sort m as the composition of $p; c$ in $\mathbf{C}$.

Hereafter we will use $c_{\mathfrak{X}}$ to denote the operation c of the algebra $\mathfrak{X}$, and c to mean both the operation $c_{\widehat{\mathbf{C}}}$ and the arrow $c \in ||\mathbf{C}||$. Moreover we will not specify the sort of sets and operations, in order to make the whole presentation more readable.

Definition 9. *The functor $\mathbf{F} : \mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})} \to \mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$ is defined as follows.
For each $\mathfrak{X} = \langle X, a_{\mathfrak{X}}, b_{\mathfrak{X}}, \dots\rangle \in \mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$, $\mathbf{F}(\mathfrak{X}) = \langle \mathbf{P_C}(X), a_{\mathbf{F}(\mathfrak{X})}, b_{\mathbf{F}(\mathfrak{X})}, \dots\rangle$ where $\forall a \in \Gamma(\mathcal{R})$, $\forall A \in \mathbf{P_C}(X)$, $a_{\mathbf{F}(\mathfrak{X})}(A) = \{(c, d_{\mathfrak{X}}(x))$ s.t. diagram (ii) in Fig. 1 commutes in $\mathbf{C}$, $d \in \mathbf{D}$ and $(b, x) \in A\}$. On arrows of $\mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$ is defined as $\mathbf{P_C}$.*

Trivially $\mathbf{F}$ is a lifting of $\mathbf{P_C}$. Then, by Prop. 3, $\mathbf{Coalg_F}$ is a category of bialgebras, it has final object $1_{\mathbf{Coalg_F}}$ and bisimilarity abstracts away from the algebraic structure.

In [19], Turi and Plotkin show that every process algebra whose operational semantics is given by SOS rules in DeSimone format, defines a bialgebra. In that approach the carrier of the bialgebra is an initial algebra T_Σ for a given algebraic signature Σ, and the SOS rules in DeSimone format specify how an endofunctor $\mathbf{F_\Sigma}$ behaves with respect to the operations of the signature. Since there exists only one arrow $?_\Sigma : T_\Sigma \to \mathbf{F_\Sigma}(T_\Sigma)$, to give the SOS rules is enough for defining a bialgebra (i.e., $\langle T_\Sigma, ?_\Sigma\rangle$) and then for assuring compositionality of bisimilarity. Our construction slightly differs from this. Indeed, the carrier of our coalgebra is $\widehat{\mathbf{C}}$, that is not the initial algebra of $\mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$. Then there could exist several or no structured coalgebras with carrier $\widehat{\mathbf{C}}$. In the following we prove that $\alpha_{\mathcal{R}} : \widehat{\mathbf{C}} \to \mathbf{F}(\widehat{\mathbf{C}})$ is a $\Gamma(\mathcal{R})$-homomorphism. This automatically assures that $\langle \widehat{\mathbf{C}}, \alpha_{\mathcal{R}} \rangle$ is a structured coalgebra and then bisimilarity is a congruence with respect to the operations of $\Gamma(\mathcal{R})$.

Theorem 1. *Let $\mathcal{R} = \langle \mathbf{C}, 0, \mathbf{D}, \mathbb{R} \rangle$ be a reactive system. If $||\mathbf{C}||$ is countable and $\mathbf{C}$ has strict distinguished object, then $\langle \widehat{\mathbf{C}}, \alpha_{\mathcal{R}} \rangle$ is a $\mathbf{F}$-coalgebra.*

From the above theorem immediately follows the characterization of $\sim_{SAT}$ as final semantics. Indeed the unique cohomorphism $!_{\mathcal{R}} : \langle \widehat{\mathbf{C}}, \alpha_{\mathcal{R}} \rangle \to 1_{\mathbf{Coalg_F}}$ identifies all the bisimilar states of $\widehat{\mathbf{C}}$. In other words, for all $f, g \in ||\mathbf{C}||$ with domain 0, $f \sim_{SAT} g$ if and only if $!_{\mathcal{R}}(f) = !_{\mathcal{R}}(g)$.

5 Coalgebraic Models of *ITS*s

Analogously to the previous section, we define a $\mathbf{P_C}$-coalgebra that coincides with *ITS*.

Definition 10. *Given a reactive system $\mathcal{R} = \langle \mathbf{C}, 0, \mathbf{D}, \mathbb{R} \rangle$, the $\mathbf{P_C}$-coalgebra corresponding to its ITS is $\langle \bar{\mathbf{C}}, \alpha^I_{\mathcal{R}} \rangle$ where $\alpha^I_{\mathcal{R}}(p) = \{(c, r; d)$ s.t. diagram (i) in Fig. 1 is an IPO and $d \in \mathbf{D}$ and $\langle l, r \rangle \in \mathbb{R}\}$.*

Now we would like to lift this coalgebra to the structured setting of $\mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$, but this is impossible, since $\alpha^I_{\mathcal{R}} : \widehat{\mathbf{C}} \to \mathbf{F}(\widehat{\mathbf{C}})$ is not a $\Gamma(\mathcal{R})$-homomorphism. Then we define below a different functor that is suitable for lifting $\alpha^I_{\mathcal{R}}$.

Definition 11. *The functor $\mathbf{I} : \mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})} \to \mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$ is defined as follows.
For each $\mathfrak{X} = \langle X, a_{\mathfrak{X}}, b_{\mathfrak{X}}, \dots \rangle \in \mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$, $\mathbf{I}(\mathfrak{X}) = \langle \mathbf{P_C}(X), a_{\mathbf{I}(\mathfrak{X})}, b_{\mathbf{I}(\mathfrak{X})}, \dots \rangle$ where $\forall a \in \Gamma(\mathcal{R})$, $\forall A \in \mathbf{P_C}(X)$, $a_{\mathbf{I}(\mathfrak{X})}(A) = \{(c, d_{\mathfrak{X}}(x))$ s.t. diagram (ii) in Fig. 1 is an IPO in $\mathbf{C}$ and $(b, x) \in A$ and $d \in \mathbf{D}\}$. On arrows of $\mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$ is defined as $\mathbf{P_C}$.*

Trivially, also $\mathbf{I}$ is a lifting of $\mathbf{P_C}$. The following theorem assures that $\sim_{IPO}$ is a congruence, and gives us a characterization as final semantics.

Theorem 2. *Let $\mathcal{R} = \langle \mathbf{C}, 0, \mathbf{D}, \mathbb{R} \rangle$ be a reactive system with redex-RPOs. If $||\mathbf{C}||$ is a countable and $\mathbf{C}$ has strict distinguished object, then $\langle \widehat{\mathbf{C}}, \alpha^I_{\mathcal{R}} \rangle$ is an $\mathbf{I}$-coalgebra.*

It is worth to note that the existence of redex-RPOs is fundamental in order to prove that $\alpha^I_{\mathcal{R}} : \widehat{\mathbf{C}} \to \mathbf{I}(\widehat{\mathbf{C}})$ is a $\Gamma(\mathcal{R})$-homomorphism, while it is not necessary for $\alpha_{\mathcal{R}}$.

A different coalgebraic construction for *ITS* has been already proposed in [5].

6 Normalized Coalgebras

The coalgebraic characterization of $\sim_{SAT}$ given in Sec. 4, is not completely satisfactory. While it supplies a characterization as final semantics, it does not allow for a minimization procedure because *CTS* is usually infinitely branching. Similar motivations have driven us to introduce semi-saturated bisimilarity in [4], that efficiently characterizes saturated bisimilarity. In this section we use the main intuition of semi-saturated bisimilarity in order to give an efficient and coalgebraic characterization of $\sim_{SAT}$. We introduce $\mathbf{Coalg_N}$, the category of Normalized Coalgebras, and we prove that it is isomorphic to $\mathbf{Coalg_F}$ (Sec. 6.1). This allows us to characterizes $\sim_{SAT}$ as the final morphism in $\mathbf{Coalg_N}$. Sec. 6.2 shows that, in the finite case, the coalgebraic minimization algorithm in $\mathbf{Coalg_N}$ is computable and Ex. 2 applies it to the open net $\mathcal{N}$.

Recall the definition of semi-saturated bisimulation (Def. 5). When p proposes a move labeled by a context c, then q must answer with a move labeled by the same context, or by a smaller one. Suppose that it answers with a smaller context d. Since bisimulations are symmetric, q will propose the d move and now p must perform a transition labeled by d, or by a smaller context. Our intuition is that, if the category of contexts is in some sense well formed, and if p and q are bisimilar, at the end p and q must perform both a transition labeled with the same minimal context. All the other bigger transitions are *redundant*, i.e., meaningless in the bisimulation game.

Thus, in order to capture the right bisimilarity, we have to forget about all the redundant transitions, i.e., all transitions $p \xrightarrow{c} p'$ such that $p \xrightarrow{d} p''$ and $\exists e \in \mathbf{D}$ with $c = d; e$ and $p' \sim p''; e$. As an example consider the *ITS* of the Open Petri net $\mathcal{N}$ (Fig. 2). The transition $e \xrightarrow{y} fy$ is redundant because $e \xrightarrow{id} f$ and clearly $fy \sim fy$ (note that in our example $m; n = mn$ for all multisets m and n). The transition $a \xrightarrow{xy} e$ is redundant because $a \xrightarrow{y} c$ and $cx \sim e$ (proved in Ex. 1).

But immediately a problem arises. How can we decide which transitions are redundant, if redundancy itself depends on bisimilarity?

Our proposal is the following. First we consider redundant only the transitions $p \xrightarrow{c} p'$ such that $p \xrightarrow{d} p''$ and $p' = p''; e$ (where as usual $c = d; e$). In our example $e \xrightarrow{y} fy$ is redundant, while $a \xrightarrow{xy} e$ is not. Then we define a category of coalgebras without redundant transitions. Since in the final object, all the bisimilar states are identified, all the transitions $p \xrightarrow{c} p'$ such that $p \xrightarrow{d} p''$ and $p' \sim p''; e$ will be forgotten.

We can better understand this idea thinking about minimization. We *normalize*, i.e., we junk away all the redundant transitions (those where $p' = p''; e$) and then we minimize w.r.t. bisimilarity. Now the bisimilar states are identified and if we normalize again, we will junk away new redundant transitions. We repeat this procedure until we reach a fix point (the final object). Since all the bisimilar states are identified in the final object, we will have forgotten not only all the transitions $p \xrightarrow{c} p'$ such that $p \xrightarrow{d} p''$ and $p' = p''; e$, but also those where $p' \sim p''; e$.

Consider for example the *ITS* derived from $\mathcal{N}$ (Fig. 2(ii)). After normalization the transition $e \xrightarrow{y} fy$ disappears (Fig. 3(i)) and after minimization $e = cx$ (Fig. 3(ii)). If we normalize again we also junk away the transition $a \xrightarrow{xy} e$ and performing a further minimization we reach the LTS depicted in Fig. 3(iii).

It is worth to note that normalization and minimization have to be repeated iteratively. Indeed we cannot minimize once and then normalize, or normalize once and then minimize (try with our example).

Definition 12 (Normalized Set and Normalization). *Let $\mathcal{R} = \langle \mathbf{C}, 0, \mathbf{D}, \mathbb{R} \rangle$ be a reactive system. Let $\mathfrak{X}$ be a $\Gamma(\mathcal{R})$-algebra with carrier set X and $A \in \mathbf{P_C}(X)$.*

A transition (c', x') derives *(c, x) in $\mathfrak{X}$ (in symbols $(c', x') \vdash_{\mathfrak{X}} (c, x)$) iff $\exists d \in \mathbf{D}$ such that $c'; d = c$ and $d_{\mathfrak{X}}(x') = x$. A transition (c', x') is* equivalent to *(c, x) in $\mathfrak{X}$*

$$\begin{tikzcd}
b \arrow[r, "y"] & c \arrow[r, "x"] & d \\
 & cx \arrow[ur, "id"] & \\
a \arrow[uur, "y"] \arrow[r, "xy"] & e \arrow[r, "id"] & f
\end{tikzcd}
\qquad
\begin{tikzcd}
b \arrow[r, "y"] & c \arrow[r, "x"] & d = f \\
a \arrow[ur, "y"] \arrow[r, "xy"] & e = cx \arrow[ur, "id"] &
\end{tikzcd}
\qquad
\begin{tikzcd}
a = b \arrow[r, "y"] & c \arrow[r, "x"] & d = f \\
 & cx = e \arrow[ur, "id"] &
\end{tikzcd}$$

(i) (ii) (iii)

Fig. 3. (i) The portion of $\langle \widehat{\mathbf{OPL}}, \alpha^I_{\mathcal{N}}; norm_{\widehat{\mathbf{OPL}}} \rangle$ corresponding to a and b.(ii) $\langle \mathfrak{B}, \beta \rangle$ is not a $\mathbf{N}$-coalgebra. (iii)$\langle \mathfrak{C}, \gamma \rangle$ is a $\mathbf{N}$-coalgebra.

($(c', x') \equiv_{\mathfrak{X}} (c, x)$) iff $(c', x') \vdash_{\mathfrak{X}} (c, x)$ and $(c, x) \vdash_{\mathfrak{X}} (c', x')$. A transition (c', x') dominates *(c, x) in $\mathfrak{X}$ ($(c', x') \prec_{\mathfrak{X}} (c, x)$) iff $(c', x') \vdash_{\mathfrak{X}} (c, x)$ and $(c, x) \nvdash_{\mathfrak{X}} (c', x')$. A transition $(c, x) \in A$ is said* redundant in *A w.r.t. $\mathfrak{X}$ if $\exists (c', x') \in A$ such that $(c', x') \prec_{\mathfrak{X}} (c, x)$.*

A is normalized *in $\mathfrak{X}$ iff it does not contain redundant transitions and it is closed by equivalent transitions. The set $\mathbf{P}^{\mathbf{N}\mathfrak{X}}_{\mathbf{C}}(X)$ is the subset of $\mathbf{P}_{\mathbf{C}}(X)$ containing all and only the normalized sets in $\mathfrak{X}$.*

For any $A \in \mathbf{P}_{\mathbf{C}}(X)$, the normalization function *$norm_{\mathfrak{X}} : \mathbf{P}_{\mathbf{C}}(X) \to \mathbf{P}^{\mathbf{N}\mathfrak{X}}_{\mathbf{C}}(X)$ maps $A \in \mathbf{P}_{\mathbf{C}}(X)$ in $\{(c', l')$ s.t. $(c', l') \equiv (c, l) \in A$ and (c, l) not redundant in A w.r.t. $\mathfrak{X}\}$.*

Look at the *ITS* of $\mathcal{N}$ in Fig.2(ii). The set of IPO transitions of e (i.e., $\alpha^I_{\mathcal{N}}(e)$) is not normalized in $\widehat{\mathbf{OPL}}$, because $(id, f) \prec_{\widehat{\mathbf{OPL}}} (x, fx)$, while the set of IPO transitions of a is normalized since $(x, c) \nvdash_{\widehat{\mathbf{OPL}}} (xy, e)$ because $y_{\widehat{\mathbf{OPL}}}(c) \neq e$. (Remember that $\mathbf{OPL}$ is the base category of $\mathcal{N}$. The algebra $\widehat{\mathbf{OPL}}$ can be thought roughly as an algebra having multisets as both elements and operators where $\forall m, n$ multisets, $m(n) = m \oplus n$ where $\oplus$ is the union of multisets).

Normalizing a set of transitions means eliminating all the redundant transitions and then closing w.r.t. $\equiv$. It is worth to note that we use $\prec_{\mathfrak{X}}$ (and not $\vdash_{\mathfrak{X}}$) to define redundant transitions. Indeed, suppose that $(c, x) \equiv (c', x')$ and no other transition dominates them. If we consider both redundant, then normalization erases both of them. This is in contrast with our main intuition of normalization, i.e., the normalized set must contain all the minimal transitions needed to derive the original set (Lemma 1).

Definition 13 (Normalizable Reactive System). *A Reactive System $\mathcal{R} = \langle \mathbf{C}, 0, \mathbf{D}, \mathbb{R} \rangle$ is normalizable if:*

1. *$||\mathbf{C}||$ is countable,*
2. *$\mathbf{C}$ has strict distinguished object,*
3. *$\forall \mathfrak{X} \in \mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$, $\prec_{\mathfrak{X}}$ is well founded.*

We inherit the first and the second constraint by Sec. 4. The third assures that for any transition, there exists a minimal non redundant transition that dominates it.

Lemma 1. *Let $\mathcal{R}$ be a normalizable reactive system. Let $\mathfrak{X}$ be $\Gamma(\mathcal{R})$-algebra and $A \in |\mathbf{F}(\mathfrak{X})|$. Then $\forall (d, x) \in A$, $\exists (d', x') \in norm_{\mathfrak{X}}(A)$, such that $(d', x') \prec_{\mathfrak{X}} (d, x)$.*

We cannot prove that the third constraint is less restrictive than requiring to have redex-RPOs, but while the latter usually does not hold in categories representing syntactic contexts (look at Ex. 2.2.2. of [18]), the former will usually hold. Indeed it just requires that a context cannot be decomposed infinitely many times.

Definition 14. *The functor* $\mathbf{N} : \mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})} \to \mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$ *is defined as follows. For each* $\mathfrak{X} = \langle X, a_{\mathfrak{X}}, b_{\mathfrak{X}}, \dots \rangle$, $\mathbf{N}(\mathfrak{X}) = \langle \mathbf{P}_{\mathbf{C}}^{\mathbf{N}\mathfrak{X}}(X), a_{\mathbf{F}(\mathfrak{X})}; norm_{\mathfrak{X}}, b_{\mathbf{F}(\mathfrak{X})}; norm_{\mathfrak{X}}, \dots \rangle$.
For all $h : \mathfrak{X} \to \mathfrak{Y}$, $\mathbf{N}(h) = \mathbf{F}(h); norm_{\mathfrak{Y}}$.

The $\mathbf{I}$-coalgebra corresponding to $\mathcal{N}$, namely $\langle \widehat{\mathbf{OPL}}, \alpha_{\mathcal{N}}^{I} \rangle$ (partially in Fig.2(ii)), is not a $\mathbf{N}$-coalgebra since $\alpha_{\mathcal{N}}^{I}(e)$ is not normalized in $\widehat{\mathbf{OPL}}$. On the other hand, it is easy too see that $\langle \widehat{\mathbf{OPL}}, \alpha_{\mathcal{N}}^{I}; norm_{\widehat{\mathbf{OPL}}} \rangle$ (partially represented in Fig.3(i)) is a $\mathbf{N}$-coalgebra.

Note how the functor is defined on arrows. If we apply $\mathbf{F}(h)$ to a normalized set A, the resulting set may not be normalized. Thus we apply the normalization function $norm_{\mathfrak{Y}}$, after the mapping $\mathbf{F}(h)$.

This is the most important intuition behind normalized coalgebras. Normalization after mapping makes bisimilar also transition systems which are such only forgetting redundant transitions. Let $\mathfrak{C}$ be the algebra obtained by quotienting $\widehat{\mathbf{OPL}}$ with $e = xc$, $a = b$ and $d = f$ and let h be such a quotient. Let γ be the transition structure on $\mathfrak{C}$ partially represented in Fig.3(iii). We have that $\alpha_{\mathcal{N}}^{I}; norm_{\widehat{\mathbf{OPL}}}; \mathbf{F}(h) \neq h; \gamma$, since $\alpha_{\mathcal{N}}^{I}; norm_{\widehat{\mathbf{OPL}}}; \mathbf{F}(h)(a) = \{(xy, e), (y, c)\}$ and $h; \gamma(a) = \{(y, c)\}$. But $\alpha_{\mathcal{N}}^{I}; norm_{\widehat{\mathbf{OPL}}}; \mathbf{F}(h); norm_{\mathfrak{Y}} = h; \gamma$ ($\alpha_{\mathcal{N}}^{I}; norm_{\widehat{\mathbf{OPL}}}; \mathbf{F}(h); norm_{\mathfrak{Y}}(a) = \{(y, c)\}$).

Now we would like to apply the theory illustrated in Sec. 3, as we have done for $\mathbf{F}$ and $\mathbf{I}$, but this is impossible since the notion of normalization (and hence the functor) strictly depends on the algebraic structure. In categorical terms, this means that $\mathbf{N}$-coalgebras are not bialgebras, or equivalently, that there exists no functor $\mathbf{B} : \mathbf{Set}^{|\mathbf{S}|} \to \mathbf{Set}^{|\mathbf{S}|}$ such that $\mathbf{N}$ is a lifting of $\mathbf{B}$.

6.1 Isomorphism Theorem

Here we prove that $\mathbf{Coalg}_{\mathbf{F}}$ and $\mathbf{Coalg}_{\mathbf{N}}$ are isomorphic. This assures that $\mathbf{Coalg}_{\mathbf{N}}$ has a final object. Moreover the final morphism in $\mathbf{Coalg}_{\mathbf{N}}$ still characterizes $\sim_{SAT}$.

We start by introducing a new category of coalgebras that is isomorphic to both $\mathbf{Coalg}_{\mathbf{F}}$ and $\mathbf{Coalg}_{\mathbf{N}}$.

Definition 15. *Let* $\mathcal{R}$ *be a reactive system and* $\mathfrak{X}$ *be a* $\Gamma(\mathcal{R})$*-algebra with carrier set* X*. A set* $A \in \mathbf{P_C}(X)$ *is* saturated *in* $\mathfrak{X}$ *if and only if it is closed w.r.t.* $\vdash_{\mathfrak{X}}$*. The set* $\mathbf{P}_{\mathbf{C}}^{\mathbf{S}\mathfrak{X}}(X)$ *is the subset of* $\mathbf{P_C}(X)$ *containing all and only the saturated sets in* $\mathfrak{X}$*.*

For any $A \in \mathbf{P_C}(X)$*, the* saturation function $sat_{\mathfrak{X}} : \mathbf{P_C}(X) \to \mathbf{P}_{\mathbf{C}}^{\mathbf{S}\mathfrak{X}}(X)$ *maps* A *to* $\{(c', x')$ *s.t.* $(c, x) \in A$ *and* $(c, x) \vdash_{\mathfrak{X}} (c', x')\}$.

The functor $\mathbf{S} : \mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})} \to \mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$ *is defined as follows.*
For each $\mathfrak{X} = \langle X, a_{\mathfrak{X}}, b_{\mathfrak{X}}, \dots \rangle \in \mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$, $\mathbf{S}(\mathfrak{X}) = \langle \mathbf{P}_{\mathbf{C}}^{\mathbf{S}\mathfrak{X}}(X), a_{\mathbf{F}(\mathfrak{X})}, b_{\mathbf{F}(\mathfrak{X})}, \dots \rangle$.
On arrows of $\mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$ *is defined as* $\mathbf{F}$.

The only difference between $\mathbf{S}$ and $\mathbf{F}$ is that for any $\Gamma(\mathcal{R})$-algebra $\mathfrak{X}$, $|\mathbf{S}(\mathfrak{X})|$ contains only the saturated set of transitions, while $|\mathbf{F}(\mathfrak{X})|$ contains all the possible sets of transitions. In terms of coalgebras this means that the coalgebras of the former functor are forced to perform only saturated set of transitions, while coalgebras of the latter are not. But every $\mathbf{F}$-coalgebra $\langle \mathfrak{X}, \alpha \rangle$ is however forced to performs only saturated set of transitions. Indeed $\forall x \in |\mathfrak{X}|$, $\alpha(x) = \alpha(id_{\mathfrak{X}}(x)) = id_{\mathbf{F}(\mathfrak{X})}(\alpha(x))$ because α is a homomorphism. By definition $id_{\mathbf{F}(\mathfrak{X})}(\alpha(x))$ is closed w.r.t. $\vdash_{\mathfrak{X}}$, i.e., saturated.

The left triangle of diagram (i) in Fig.4 depicts this setting. $\mathbf{S}(\mathfrak{X})$ is a subalgebra of $\mathbf{F}(\mathfrak{X})$, i.e. $\forall \mathfrak{X} \in \mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$, $\exists m_{\mathfrak{X}} : \mathbf{S}(\mathfrak{X}) \rightarrowtail \mathbf{F}(\mathfrak{X})$ mono. Moreover $\forall \alpha : \mathfrak{X} \to \mathbf{F}(\mathfrak{X})$, there exists a unique $\alpha_S : \mathfrak{X} \to \mathbf{S}(\mathfrak{X})$ such that $\alpha = \alpha_S; m_{\mathfrak{X}}$. This observation guarantees that $\mathbf{Coalg_F}$ and $\mathbf{Coalg_S}$ are isomorphic. While, in order to prove the isomorphism of $\mathbf{Coalg_S}$ and $\mathbf{Coalg_N}$, we show that normalization and saturation are natural isomorphisms.

Proposition 4. *Let norm and sat be respectively the families of morphisms* $\{norm_{\mathfrak{X}} : \mathbf{S}(\mathfrak{X}) \to \mathbf{N}(\mathfrak{X})\ \forall \mathfrak{X} \in |\mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}|\}$ *and* $\{sat_{\mathfrak{X}} : \mathbf{N}(\mathfrak{X}) \to \mathbf{S}(\mathfrak{X})\ \forall \mathfrak{X} \in |\mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}|\}$. *Then* $norm : \mathbf{S} \Rightarrow \mathbf{N}$ *and* $sat : \mathbf{N} \Rightarrow \mathbf{S}$ *are natural transformations, and one is the inverse of the other, i.e. all squares in diagram (ii) of Fig.4 commutes.*

Theorem 3. $\mathbf{Coalg_F}$, $\mathbf{Coalg_S}$ *and* $\mathbf{Coalg_N}$ *are isomorphic.*

$$\begin{tikzcd}
& (\mathfrak{X}) \arrow[dl, "\alpha"'] \arrow[rr, "h"] \arrow[dd, dotted, "\alpha_S"] & & (\mathfrak{Y}) \arrow[dl, "\beta"'] \arrow[dd, dotted, "\beta_S"] \\
\mathbf{F}(\mathfrak{X}) \arrow[rr, "\mathbf{F}(h)"] & & \mathbf{F}(\mathfrak{Y}) & \\
& \mathbf{S}(\mathfrak{X}) \arrow[ul, tail, "m_{\mathfrak{X}}"] \arrow[rr, "\mathbf{S}(h)"'] & & \mathbf{S}(\mathfrak{Y}) \arrow[ul, tail, "m_{\mathfrak{Y}}"']
\end{tikzcd}$$

(i)

$$\begin{tikzcd}
\mathbf{S}(\mathfrak{X}) \arrow[loop above, "id_{\mathbf{S}(\mathfrak{X})}"] \arrow[r, "\mathbf{S}(h)"] \arrow[d, bend right, "norm_{\mathfrak{X}}"'] & \mathbf{S}(\mathfrak{Y}) \arrow[loop above, "id_{\mathbf{S}(\mathfrak{Y})}"] \arrow[d, bend right, "norm_{\mathfrak{Y}}"'] \\
\mathbf{N}(\mathfrak{X}) \arrow[loop below, "id_{\mathbf{N}(\mathfrak{X})}"] \arrow[r, "\mathbf{N}(h)"'] \arrow[u, bend right, "sat_{\mathfrak{X}}"'] & \mathbf{N}(\mathfrak{Y}) \arrow[loop below, "id_{\mathbf{N}(\mathfrak{Y})}"] \arrow[u, bend right, "sat_{\mathfrak{Y}}"']
\end{tikzcd}$$

(ii)

Fig. 4. Commuting diagrams in $\mathbf{Alg}_{\mathbf{\Gamma}(\mathcal{R})}$

The above theorem guarantees that $\mathbf{Coalg_N}$ has a final system $1_{\mathbf{Coalg_N}}$. The final morphisms $!^N_{\mathcal{R}} : \langle \widehat{\mathbf{C}}, \alpha_{\mathcal{R}}; norm_{\widehat{\mathbf{C}}} \rangle \to 1_{\mathbf{Coalg_N}}$ characterizes $\sim_{SAT}$.

Corollary 1. *Let* $\mathcal{R}$ *be a normalizable reactive system.* $p \sim_{SAT} q \Leftrightarrow !_{\mathcal{R}}(p) = !_{\mathcal{R}}(q) \Leftrightarrow !^N_{\mathcal{R}}(p) = !^N_{\mathcal{R}}(q)$.

6.2 From *ITS* to $\sim_{SAT}$ Through Normalization

Until now, we have proved that $!^N_{\mathcal{R}} : \langle \widehat{\mathbf{C}}, \alpha_{\mathcal{R}}; norm_{\widehat{\mathbf{C}}} \rangle \to 1_{\mathbf{Coalg_N}}$ characterizes $\sim_{SAT}$. Now we apply the coalgebraic minimization algorithm (Sec. 3) in the category $\mathbf{Coalg_N}$, in order to compute $\sim_{SAT}$. However, normalizing $\alpha_{\mathcal{R}}$ is unfeasible, because it is usually infinitely branching. Instead of normalizing the *CTS*, we can build $\alpha_{\mathcal{R}}; norm_{\widehat{\mathbf{C}}}$ through the normalization of *ITS*.

Note that the *ITS* could have redundant transitions. Indeed consider two redex squares for two different rules as those depicted in diagram (vi) of Fig. 1 where $\langle l, r \rangle$,

$\langle l', r' \rangle \in \mathbb{R}$. The transition $p \xrightarrow{c} r; d$ could be an IPO transition even if it is dominated by $p \xrightarrow{c'} r'; d'$. This explains the difference between our notion of redundancy and that of Leifer and Milner. They consider all the non-IPO transitions redundant, i.e. all the transitions where the label contains something that is not strictly necessary to reach the rule. Our notion completely abstracts from rules.

Theorem 4. *Let $\mathcal{R} = \langle \mathbf{C}, 0, \mathbf{D}, \mathbb{R} \rangle$ be a normalizable reactive system having IPOs. Then $\alpha^I_{\mathcal{R}}; norm_{\widehat{\mathbf{C}}} = \alpha_{\mathcal{R}}; norm_{\widehat{\mathbf{C}}}$ and moreover, $a_{\mathbf{I}(\mathfrak{X})}; norm_{\mathfrak{X}} = a_{\mathbf{F}(\mathfrak{X})}; norm_{\mathfrak{X}}$.*

This theorem is the key to compute $!^N_{\mathcal{R}}(p)$. Indeed it allows to compute $\alpha^I_{\mathcal{R}}; norm_{\widehat{\mathbf{C}}}$ instead of $\alpha_{\mathcal{R}}; norm_{\widehat{\mathbf{C}}}$ that is usually unfeasible. Now we can instantiate the general minimization algorithm (Sec. 3) in the case of $\mathbf{Coalg_N}$.

At the beginning $!^N_0 : \widehat{\mathbf{C}} \to \mathbb{1}$ is the final morphism to $\mathbb{1}$ (the final $\Gamma(\mathcal{R})$-algebra).

At any iteration $!^N_{n+1} = \alpha_{\mathcal{R}}; norm_{\widehat{\mathbf{C}}}; \mathbf{N}(!^N_n) = \alpha^I_{\mathcal{R}}; norm_{\widehat{\mathbf{C}}}; \mathbf{F}(!^N_n); norm_{\mathbf{N}(\mathbb{1})^n}$.

The peculiarity of minimization in $\mathbf{Coalg_N}$ is that we must normalize at every iteration. Note that the normalization is performed not only in the source algebra $\widehat{\mathbf{C}}$, but also on the target algebra $\mathbf{N}(\mathbb{1})^n$. Thus the minimization procedure strictly depends on the algebraic structure. This further explains why normalized coalgebras are structured coalgebras but not bialgebras where we can completely forget about the algebraic structure.

Proposition 5. *Let $\mathcal{R} = \langle \mathbf{C}, 0, \mathbf{D}, \mathbb{R} \rangle$ be a normalizable reactive system such that:*

- *arrow composition in $\mathbf{C}$ is computable,*
- *$\mathbf{C}$ has IPOs, and these can be computed,*
- *$\forall a, b \in ||\mathbf{C}||$, there exist a finite number of $c \in ||\mathbf{C}||$ such that $a = b; c$,*
- *$\forall a, b \in ||\mathbf{C}||$, there exist a finite number of $c, d \in ||\mathbf{C}||$ such that diagram (ii) in Fig. 1 is an IPO.*

Then the algorithm outlined above is computable and it terminates for minimizing those p whose ITS is finite.

Example 2. Here we prove that $a \sim_{SAT} b$ in $\mathcal{N}$ (Ex. 1), by proving that $!^N_{\mathcal{N}}(a) = !^N_{\mathcal{N}}(b)$. The LTSs $\alpha_{\mathcal{N}}; norm_{\widehat{\mathbf{OPL}}}(a)$ and $\alpha_{\mathcal{N}}; norm_{\widehat{\mathbf{OPL}}}(b)$ are shown in Fig.3(i) (by Th. 4 these can be computed by normalizing in $\widehat{\mathbf{OPL}}$ the *ITS* in Fig. 2(ii)). Let $\Gamma(\mathcal{N})$ be the specification corresponding to $\mathcal{N}$: operations are just multisets on $\{x, y\}$. Let $\mathbb{1}$ be the final $\Gamma(\mathcal{N})$-algebra: the carrier set contains only the single element 1 and for all operations m, $m(1) = 1$.

The homomorphism $!^N_0 : \widehat{\mathbf{OPL}} \to \mathbb{1}$ maps all the elements of $|\widehat{\mathbf{OPL}}|$ into 1.

In order to compute $!^N_1$, we first compute $\alpha_{\mathcal{N}}; norm_{\widehat{\mathbf{OPL}}}; \mathbf{F}(!^0)$ for all the states reachable from a and b (the results are reported in the second column of Fig. 5(i)) and then we normalize in the final algebra $\mathbb{1}$ (third column). The normalization junks away the transition $a \xrightarrow{xy} 1$. Indeed $(y, 1) \prec_{\mathbb{1}} (xy, 1)$ since $xy = y; x$ and $x_{\mathbb{1}}(1) = 1$.

For computing $!^N_2$ we proceed as before, using $!^N_1$ instead of $!^N_0$ and normalizing on $\mathbf{N}(\mathbb{1})$ instead of normalizing on $\mathbb{1}$. The results of the second iteration are reported in Fig. 5(ii). Normalization junks away the transitions $a \xrightarrow{xy} \{(id, 1)\}$ because $(y, \{(x, 1)\}) \prec_{\mathbf{N}(\mathbb{1})} (xy, \{(id, 1)\})$. The morphism $!^N_2$ partitions the states in $\{a, b\}$, $\{c\}$, $\{d, f\}$, $\{e\}$, as well as $!^N_1$. Thus the algorithm terminates and then $a \sim_{SAT} b$.

multisets	$\alpha_{\mathcal{N}}; norm_{\widehat{\mathbf{OPL}}}; \mathbf{F}(!_0^N)$	$!_1^N$
a	$(xy,1),(y,1)$	$(y,1)$
b	$(y,1)$	$(y,1)$
c	$(x,1)$	$(x,1)$
d	$\varnothing$	$\varnothing$
e	$(id,1)$	$(id,1)$
f	$\varnothing$	$\varnothing$

(i)

multisets	$\alpha_{\mathcal{N}}; norm_{\widehat{\mathbf{OPL}}}; \mathbf{F}(!_1^N)$	$!_2^N$
a	$(xy,\{(id,1)\}),(y,\{(x,1)\})$	$(y,\{(x,1)\})$
b	$(y,\{(x,1)\})$	$(y,\{(x,1)\})$
c	$(x,\varnothing)$	$(x,\varnothing)$
d	$\varnothing$	$\varnothing$
e	$(id,\varnothing)$	$(id,\varnothing)$
f	$\varnothing$	$\varnothing$

(ii)

Fig. 5. (i) First Iteration. (ii)Second Iteration.

7 Conclusions

In this paper we have defined structured coalgebras for both labeled transition systems (*CTS* and *ITS*), derived from reactive systems. This provides a characterization of $\sim_{SAT}$ and $\sim_{IPO}$ via final semantics and minimal realizations. Since *CTS* is usually infinite branching, its minimal realization is infinite and $\sim_{SAT}$ uncomputable. For this reason, we have introduced Normalized Coalgebras. These are structured coalgebras that, thanks to a suitable definition of the underlying functor, allow to forget about redundant transitions but still characterize $\sim_{SAT}$ as final semantics. Here the notion of redundancy is coarser than that expressed by the IPO condition. Indeed, given $p \xrightarrow{y} q$, $p \xrightarrow{x} q'$ and a context $E[-]$, such that $E[x] = y$, the former transition is not an IPO if the latter reacts with the same rule and $E[q'] = q$, while it is redundant, according to our notion of normalized coalgebras, if $E[q'] \sim q$ (without any condition on rules). Then constructing an LTS smaller than *ITS* and then minimizing it through a minimization algorithm (which employs the proper functor definition) allows us to check $\sim_{SAT}$. This approach can be easily extended to G-reactive systems [17] and to open reactive systems [11] where, in our opinion, it might help to relax the constraints of the theory. Pragmatically, Normalized Coalgebras are isomorphic to a category of bialgebras, but the minimization procedure is feasible, because it employs the algebraic structure that is completely forgotten in bialgebras.

References

1. Adámek, J., Koubek, V.: On the greatest fixed point of a set functor. Theoretical Computer Science 150(1) (1995)
2. Baldan, P., Corradini, A., Ehrig, H., Heckel, R.: Compositional semantics for open Petri nets based on deterministic processes. Mathematical Structures in Computer Science 15(1), 1–35 (2005)
3. Berry, G., Boudol, G.: The chemical abstract machine. Theoretical Computer Science 96, 217–248 (1992)
4. Bonchi, F., König, B., Montanari, U.: Saturated semantics for reactive systems. In: Proc. of LICS, pp. 69–80. IEEE Computer Society Press, Los Alamitos (2006)
5. Bonchi, F., Montanari, U.: A coalgebraic theory of reactive systems. Electronic Notes in Theoretical Computer Science, LIX Colloqium on Emerging Trends in Concurrency Theory (to appear)

6. Corradini, A., Große-Rhode, M., Heckel, R.: A coalgebraic presentation of structured transition systems. Theoretical Computer Science 260, 27–55 (2001)
7. Corradini, A., Heckel, R., Montanari, U.: From sos specifications to structured coalgebras: How to make bisimulation a congruence. Electronic Notes in Theoretical Computer Science, vol. 19 (1999)
8. Ehrig, H., König, B.: Deriving bisimulation congruences in the DPO approach to graph rewriting. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 151–166. Springer, Heidelberg (2005)
9. Kanellakis, P.C., Smolka, S.A.: Ccs expressions, finite state processes, and three problems of equivalence. Information and Computation 86(1), 43–68 (1990)
10. Kindler, E.: A compositional partial order semantics for Petri net components. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 235–252. Springer, Heidelberg (1997)
11. Klin, B., Sassone, V., Sobocinski, P.: Labels from reductions: Towards a general theory. In: Fiadeiro, J.L., Harman, N., Roggenbach, M., Rutten, J. (eds.) CALCO 2005. LNCS, vol. 3629, pp. 30–50. Springer, Heidelberg (2005)
12. Leifer, J.J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 243–258. Springer, Heidelberg (2000)
13. Milner, R.: Bigraphs for Petri nets. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 686–701. Springer, Heidelberg (2004)
14. Milner, R.: Bigraphical reactive systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 16–35. Springer, Heidelberg (2001)
15. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249(1), 3–80 (2000)
16. Sassone, V., Sobociński, P.: A congruence for Petri nets. In: Ehrig, H., Padberg, J., Rozenberg, G. (eds.) Petri Nets and Graph Transformation. ENTCS, vol. 127, pp. 107–120. Elsevier, Amsterdam (2005)
17. Sassone, V., Sobociński, P.: Reactive systems over cospans. In: Proc. of LICS, pp. 311–320. IEEE Computer Society Press, Los Alamitos (2005)
18. Sobociński, P.: Deriving process congruences from reaction rules. PhD thesis (2004)
19. Turi, D., Plotkin, G.D.: Towards a mathematical operational semantics. In: Proc. of LICS, pp. 280–291. IEEE Computer Society Press, Los Alamitos (1997)

Reactive Systems over Directed Bigraphs*

Davide Grohmann and Marino Miculan

Dept. of Mathematics and Computer Science, University of Udine, Italy
grohmann@dimi.uniud.it, miculan@dimi.uniud.it

Abstract. We study the construction of labelled transition systems from reactive systems defined over *directed bigraphs*, a computational meta-model which subsumes other variants of bigraphs. First we consider wide transition systems whose labels are all those generated by the IPO construction; the corresponding bisimulation is always a congruence. Then, we show that these LTSs can be simplified further by restricting to a subclass of labels, which can be characterized syntactically.

We apply this theory to the Fusion calculus: we give an encoding of Fusion in directed bigraphs, and describe its simplified wide transition system and corresponding bisimulation.

1 Introduction

Bigraphical reactive systems (BRSs) are an emerging graphical meta-model of concurrent computation introduced by Milner [5,6]. The key structure of BRSs are *bigraphs*, which are composed by a hierarchical *place graph* describing locations, and a *link (hyper-)graph* describing connections. The dynamics of a system is represented by a set of reaction rules which may change both these structures. Remarkably, using a general construction based on the notion of *relative pushout* (RPO), from a BRS we can obtain a labelled transition system such that the associated bisimulation is always a congruence [3].

Several process calculi for Concurrency can be represented in bigraphs, such as CCS and (using a mild generalization), also the π-calculus and the λ-calculus. Nevertheless, other calculi, such as Fusion [7], seem to escape this framework. On the other hand, a "dual" version of bigraphs introduced by Sassone and Sobociński [8] seem suitable for Fusion calculus, but not for the others.

In order to overcome this limitation, in previous work we have introduced a generalization of both Milner's and Sassone-Sobociński's variants, called *directed bigraphs* [2,1]. Intuitively, in directed bigraphs edges represent *resources* which are *accessed* by controls, and *indicated* by names. Connections from controls to edges (through names) represent "resource requests", or accesses. Directed bigraphs feature an RPO construction which generalizes and unifies both Jensen-Milner's and Sassone-Sobociński's variants [2].

In this paper, we consider reactive systems built over directed bigraphs, hence called *directed bigraphical reactive system* (DBRS). Given a DBRS, we can readily obtain a labelled transition system (called *directed bigraphical transition system*, DBTS) by taking as observations all the contexts generated by the IPO

* Work supported by Italian MIUR project 2005015824 ART.

construction. We show that the bisimilarity associated to this DBTS (called "standard") is always a congruence. However, this LTS is still quite large, and may contain transitions not strictly necessary. In fact, we show that, under a mild condition, standard bisimilarity can be characterized also by a smaller, more tractable DBTS, whose transitions are only those really relevant for the agents.

In order to check the suitability of this theory, we apply it to the Fusion calculus. We present the first encoding of the Fusion calculus as a DBRS; then, we discuss the DBTS and associated bisimilarity constructed using these techniques.

Synopsis. In Section 2 we recall some definitions about directed bigraphs. In Section 3 we introduce directed bigraphical reactive and transition systems, and show that standard bisimilarity is a congruence. In Section 4, we show that (under some conditions) standard bisimilarity can be characterized by a smaller LTS. As an application, in Section 5 we represent the Fusion Calculus using DBRSs and DBTSs. Conclusions and direction for future work are in Section 6.

2 Directed Bigraphs

In this section we recall some definitions about directed bigraphs, as in [2]. Following Milner's approach, we work in *precategories;* see [4, §3] for an introduction to the theory of supported monoidal precategories.

Let $\mathcal{K}$ be a given signature of controls, and $ar : \mathcal{K} \to \omega$ the arity function.

Definition 1. *A* polarized interface X *is a pair of disjoint sets of names* $X = (X^-, X^+)$*; the two elements are called* downward *and* upward *faces, respectively.*

A directed link graph $A : X \to Y$ *is* $A = (V, E, ctrl, link)$ *where* X *and* Y *are the* inner *and* outer *interfaces,* V *is the set of* nodes*,* E *is the set of* edges*,* $ctrl : V \to \mathcal{K}$ *is the* control map*, and* $link : \mathsf{Pnt}(A) \to \mathsf{Lnk}(A)$ *is the* link map*, where the* ports*, the* points *and the* links *of* A *are defined as follows:*

$$\mathsf{Prt}(A) \triangleq \sum_{v \in V} ar(ctrl(v)) \qquad \mathsf{Pnt}(A) \triangleq X^+ \uplus Y^- \uplus \mathsf{Prt}(A) \qquad \mathsf{Lnk}(A) \triangleq X^- \uplus Y^+ \uplus E$$

The link map cannot connect downward and upward names of the same interface, i.e., the following condition must hold: $(link(X^+) \cap X^-) \cup (link(Y^-) \cap Y^+) = \emptyset$.

Directed link graphs are graphically depicted much like ordinary link graphs, with the difference that edges are explicit objects and points and names are associated to edges (or other names) by (simple) directed arcs. This notation makes explicit the "resource request flow": ports and names in the interfaces can be associated either to locally defined resources (i.e., a local edge) or to resources available from outside the system (i.e., via an outer name).

Definition 2 ('DLG). *The precategory of* directed link graphs *has polarized interfaces as objects, and directed link graphs as morphisms.*

Given two directed link graphs $A_i = (V_i, E_i, ctrl_i, link_i) : X_i \to X_{i+1}$ $(i = 0, 1)$*, the composition* $A_1 \circ A_0 : X_0 \to X_2$ *is defined when the two link graphs have disjoint nodes and edges. In this case,* $A_1 \circ A_0 \triangleq (V, E, ctrl, link)$*, where*

$V \triangleq V_0 \uplus V_1$, $ctrl \triangleq ctrl_0 \uplus ctrl_1$, $E \triangleq E_0 \uplus E_1$ and $link : X_0^+ \uplus X_2^- \uplus P \to E \uplus X_0^- \uplus X_2^+$ is defined as follows (where $P = \mathsf{Prt}(A_0) \uplus \mathsf{Prt}(A_1)$):

$$link(p) \triangleq \begin{cases} link_0(p) & \text{if } p \in X_0^+ \uplus \mathsf{Prt}(A_0) \text{ and } link_0(p) \in E_0 \uplus X_0^- \\ link_1(x) & \text{if } p \in X_0^+ \uplus \mathsf{Prt}(A_0) \text{ and } link_0(p) = x \in X_1^+ \\ link_1(p) & \text{if } p \in X_2^- \uplus \mathsf{Prt}(A_1) \text{ and } link_1(p) \in E_1 \uplus X_2^+ \\ link_0(x) & \text{if } p \in X_2^- \uplus \mathsf{Prt}(A_1) \text{ and } link_1(p) = x \in X_1^- . \end{cases}$$

The identity link graph of X is $id_X \triangleq (\emptyset, \emptyset, \emptyset_\mathcal{K}, Id_{X^- \uplus X^+}) : X \to X$.

Definition 3. *The* support *of $A = (V, E, ctrl, link)$ is the set $|A| \triangleq V \oplus E$.*

Definition 4. *Let $A : X \to Y$ be a link graph.*

A link $l \in \mathsf{Lnk}(A)$ is idle *if it is not in the image of the link map (i.e., $l \notin link(\mathsf{Pnt}(A))$). The link graph A is* lean *if there are no idle links.*

A link l is open *if it is an inner downward name or an outer upward name (i.e., $l \in X^- \cup Y^+$); it is* closed *if it is an edge.*

A point p is open *if $link(p)$ is an open link; otherwise it is closed. Two points p_1, p_2 are* peer *if they are mapped to the same link, that is $link(p_1) = link(p_2)$.*

Definition 5. *The* tensor product $\otimes$ *in* 'DLG *is defined as follows. Given two objects X, Y, if these are pairwise disjoint then $X \otimes Y \triangleq (X^- \uplus Y^-, X^+ \uplus Y^+)$. Given two link graphs $A_i = (V_i, E_i, ctrl_i, link_i) : X_i \to Y_i$ $(i = 0, 1)$, if the tensor products of the interfaces are defined and the sets of nodes and edges are pairwise disjoint then the tensor product $A_0 \otimes A_1 : X_0 \otimes X_1 \to Y_0 \otimes Y_1$ is defined as $A_0 \otimes A_1 \triangleq (V_0 \uplus V_1, E_0 \uplus E_1, ctrl_0 \uplus ctrl_1, link_0 \uplus link_1)$.*

Finally, we can define the *directed bigraphs* as the composition of standard place graphs (see [4, §7] for definitions) and directed link graphs.

Definition 6 (directed bigraphs). *A* (bigraphical) interface *I is composed by a* width *(a finite ordinal, denoted by $width(I)$) and by a polarized interface of link graphs (i.e., a pair of finite sets of names).*

A directed bigraph *with signature $\mathcal{K}$ is $G = (V, E, ctrl, prnt, link) : I \to J$, where $I = \langle m, X \rangle$ and $J = \langle n, Y \rangle$ are its inner and outer interfaces respectively; V and E are the sets of nodes and edges respectively, and $prnt$, $ctrl$ and $link$ are the parent, control and link maps, such that $G^P \triangleq (V, ctrl, prnt) : m \to n$ is a place graph and $G^L \triangleq (V, E, ctrl, link) : X \to Y$ is a directed link graph.*

We denote G as combination of G^P and G^L by $G = \langle G^P, G^L \rangle$. In this notation, a place graph and a (directed) link graph can be put together if and only if they have the same sets of nodes and edges.

Definition 7 ('DBig). *The precategory* 'DBig *of directed bigraph with signature $\mathcal{K}$ has interfaces $I = \langle m, X \rangle$ as objects and directed bigraphs $G = \langle G^P, G^L \rangle : I \to J$ as morphisms. If $H : J \to K$ is another directed bigraph with sets of nodes and edges disjoint from V and E respectively, then their composition is defined by composing their components, i.e.: $H \circ G \triangleq \langle H^P \circ G^P, H^L \circ G^L \rangle : I \to K$.*

The identity directed bigraph of $I = \langle m, X \rangle$ is $\langle id_m, Id_{X^- \uplus X^+} \rangle : I \to I$.

A bigraph is *ground* if its inner interface is $\epsilon = \langle 0, (\emptyset, \emptyset) \rangle$; we denote by $Gr\langle I \rangle$ the set of ground bigraphs with outer interface I, i.e., $Gr\langle I \rangle = {}'\mathrm{DBig}(\epsilon, I)$.

Definition 8. *The* tensor product $\otimes$ *in* $'\mathrm{DBig}$ *is defined as follows. Given* $I = \langle m, X \rangle$ *and* $J = \langle n, Y \rangle$*, where* X *and* Y *are pairwise disjoint, then* $\langle m, X \rangle \otimes \langle n, Y \rangle \triangleq \langle m + n, (X^- \uplus Y^-, X^+ \uplus Y^+) \rangle$.

The tensor product of $G_i : I_i \to J_i$ *is defined as* $G_0 \otimes G_1 \triangleq \langle G_0^P \otimes G_1^P, G_0^L \otimes G_1^L \rangle : I_0 \otimes I_1 \to J_0 \otimes J_1$*, when the tensor products of the interfaces are defined and the sets of nodes and edges are pairwise disjoint.*

Remarkably, directed link graphs (and bigraphs) have relative pushouts and pullbacks, which can be obtained by a general construction, subsuming both Milner's and Sassone-Sobociński's variants. We refer the reader to [2].

Actually, in many situations we do not want to distinguish bigraphs differing only on the identity of nodes and edges. To this end, we introduce the category DBig of *abstract directed bigraphs*. The category DBig is constructed from $'\mathrm{DBig}$ forgetting the identity of nodes and edges and any idle edge. More precisely, abstract bigraphs are concrete bigraphs taken up-to an equivalence $\cong$.

Definition 9 (abstract directed bigraphs). *Two concrete directed bigraphs* G *and* H *are* lean-support equivalent*, written* $G \cong H$*, if they are support equivalent after removing any idle edges.*

The category DBig *of abstract directed bigraphs has the same objects as* $'\mathrm{DBig}$*, and its arrows are lean-support equivalence classes of directed bigraphs. We denote by* $\mathcal{A} : {}'\mathrm{DBig} \to \mathrm{DBig}$ *the associated quotient functor.*

We remark that DBig is a category (and not only a precategory); moreover, $\mathcal{A}$ enjoys several properties which we omit here due to lack of space; see [4].

3 Directed Bigraphical Reactive and Transition Systems

In this section we introduce wide reactive systems and wide transition systems over directed bigraphs, called *directed bigraphical reactive systems* and *directed bigraphical transition systems* respectively.

3.1 Directed Bigraphical Reactive Systems

We assume the reader familiar with wide reactive systems over premonoidal categories; see [6] for the relevant definitions.

In order to define wide reactive systems over directed bigraphs, we need to define how a parametric rule, that is a "redex-reactum" pair of bigraphs, can be instantiated. Essentially, in the application of the rule, the "holes" in the reactum must be filled with the parameters appearing in the redex. This relation can be expressed by a function mapping each site in width n of the reactum to a site in width m of the redex; notice that this allows to replace, duplicate or forget in the reactum the parameters of the redex. Formally:

Definition 10 (instantiation). *An* instantiation *ρ from (width) m to (width) n, written $\rho :: m \to n$, is determined by a function $\bar{\rho} : n \to m$. For any pair X, this function defines the map $\rho : Gr\langle m, X\rangle \to Gr\langle n, X\rangle$ as follows. Decompose $g : \langle m, X\rangle$ into $g = \omega(d_0 \otimes \cdots \otimes d_{m-1})$, with $\omega : (\emptyset, Y) \to X$ and each d_i ($i \in m$) prime and discrete. Then define:*

$$\rho(g) \triangleq \omega(e_0 \curlywedge \ldots \curlywedge e_{n-1} \curlywedge id_{(\emptyset,Y)})$$

where $e_j \doteq d_{\bar{\rho}(j)}$ for $j \in n$. This map is well defined (up to support equivalence).

If $\bar{\rho}$ is injective, surjective, bijective then the instantiation ρ is said to be affine, total *or* linear *respectively.*

If ρ is not affine then it replicates at least one of the factor d_i. Support translation is used to ensure that several copies of replicated factor have disjoint support; also outer sharing product is used because copies will share names.

Note that the names of $e_0 \curlywedge \ldots \curlywedge e_{n-1}$ may be fewer than Y, because ρ may be not total, so we add $id_{(\emptyset,Y)}$ to the previous product to ensure that the composition with ω is defined (in that composition idle names can be generated).

Proposition 1. *Wirings commute with instantiation: $\omega\rho(a) \doteq \rho(\omega a)$.*

Next, we define how to generate ground reaction rules from parametric rules.

Definition 11 (reaction rules for directed bigraphs). *A ground reaction rule is a pair (r, r'), where r and r' are ground with the same outer interface. Given a set of ground rules, the reaction relation $\longrightarrow$ over agents is the least, closed under support equivalence ($\doteq$), such that $Dr \longrightarrow Dr'$ for each active context D and each ground rule (r, r').*

A parametric reaction rule has a redex R and reactum R', and takes the form:

$$(R : I \to J, R' : I' \to J, \rho)$$

where the inner interface I and I' with widths m and m'. The third component $\rho :: m \to m'$ is an instantiation. For any X and discrete $d : I \otimes (\emptyset, X)$ the parametric rule generate the ground reaction rule:

$$((R \otimes id_{(\emptyset,X)}) \circ d, (R' \otimes id_{(\emptyset,X)}) \circ \rho(d)).$$

Definition 12 (directed bigraphical reactive system). *A* directed bigraphical reactive system *(DBRS) over $\mathcal{K}$ is the precategory $'\mathrm{DB}\textsc{ig}(\mathcal{K})$ equipped with a set*[1] *$'\mathcal{R}$ of reaction rules closed under support equivalence ($\doteq$). We denote it by $'\mathcal{D}(\mathcal{K}, '\mathcal{R})$, or simply $'\mathcal{D}$ when clear from context.*

3.2 Directed Bigraphical Transition Systems

We can prove that DBRSs are a particular instance of the generic wide reactive systems definable on a wide monoidal precategory [6, Definition 3.1]:

[1] Here we use the "tick" to indicate that elements are objects of a precategory.

Proposition 2. *Directed bigraphical reactive systems are wide reactive systems.*

This result ensures that DBRSs inherit from the theory of WRSs ([6]) the definition of transition system based on IPOs.

Definition 13 (directed bigraphical transition system). *A* transition *for a DBRS $'\mathcal{D}(\mathcal{K}, '\mathcal{R})$ is a quadruple (a, L, λ, a'), written as $a \xrightarrow{L}_{\lambda} a'$, where a and a' are ground bigraphs and there exist a ground reaction rule $(r, r') \in {'\mathcal{R}}$ and an active context D such that $La = Dr$, and $\lambda = width(D)(width(cod(r)))$ and $a' \simeq Dr'$. A transition is* minimal *if the (L, D) is an IPO for (a, r).*

A directed bigraphical transition system *(DBTS) $\mathcal{L}$ for $'\mathcal{D}$ is a pair $(\mathcal{I}, \mathcal{T})$:*

- *$\mathcal{I}$ is a set of bigraphical interfaces; the agents of $\mathcal{L}$ are the ground bigraphs with outer interface I, for $I \in \mathcal{I}$;*
- *$\mathcal{T}$ is a set of transitions whose sources and targets are agents of $\mathcal{L}$.*

The full *(resp.* standard*) transition* FT *(resp.* ST*) system consists of all interfaces, together with all (resp. all minimal) transitions.*

Definition 14 (bisimilarity). *Let $'\mathcal{D}$ be a DBRS equipped with a DBTS $\mathcal{L}$. A* simulation *(on $\mathcal{L}$) is a binary relation $\mathcal{S}$ between agents with equal interface such that if $a\mathcal{S}b$ and $a \xrightarrow{L}_{\lambda} a'$ in $\mathcal{L}$, then whenever Lb is defined there exists b' such that $b \xrightarrow{L}_{\lambda} b'$ and $a'\mathcal{S}b'$.*

A bisimulation *is a symmetric simulation. Then* bisimilarity *(on $\mathcal{L}$), denoted by $\sim_{\mathcal{L}}$, is the largest bisimulation.*

From [6, Theorem 4.6] we have the following property:

Proposition 3 (congruence of wide bisimilarity). *In any concrete DBRS equipped with the standard transition system, wide bisimilarity is a congruence.*

Now we want to transfer ST, with its congruence property, to the abstract DBRS $\mathcal{D}(\mathcal{K}, \mathcal{R})$, where DBIG($\mathcal{K}$) is the category defined by the quotient functor $\mathcal{A}$, and $\mathcal{R}$ is obtained from $'\mathcal{R}$ also by $\mathcal{A}$.

Recall that the functor $\mathcal{A}$ forgets idle edges. For this purpose, as in [6], we impose a constrain upon the reaction rules in $'\mathcal{R}$: every redex R must be lean. Then we can prove that transitions respect lean-support equivalence:

Proposition 4. *In any concrete DBRS with all redex lean, equipped with* ST*:*

1. *in every transition label L, both components are lean;*
2. *transitions respect lean-support equivalence ($\risingdotseq$). For every transition $a \xrightarrow{L}_{\lambda} a'$, if $a \risingdotseq b$ and $L \risingdotseq M$ where M is another label with $M \circ b$ defined, then there exists a transition $b \xrightarrow{M}_{\lambda} b'$ for some b' such that $a' \risingdotseq b'$.*

Now we want to transfer the congruence result of Proposition 3 to abstract DBTSs. The following result is immediate by the [6, Theorem 4.8].

Proposition 5. *Let $'\mathcal{D}(\mathcal{K}, '\mathcal{R})$ be a concrete DBRS with all redexes lean, with* ST*. Let $\mathcal{A} : {'\text{DBIG}}(\mathcal{K}) \to \text{DBIG}(\mathcal{K})$ be the lean-support equivalence functor. Then*

1. *$a \sim_{\text{ST}} b$ in $'\mathcal{D}(\mathcal{K}, '\mathcal{R})$ if and only if $\mathcal{A}(a)\mathcal{A}(\sim_{\text{ST}})\mathcal{A}(b)$ in $\mathcal{D}(\mathcal{K}, \mathcal{R})$;*
2. *bisimilarity $\mathcal{A}(\sim_{\text{ST}})$ is a congruence in $\mathcal{D}(\mathcal{K}, \mathcal{R})$.*

4 Reducing Directed Bigraphical Transition Systems

The standard DBTS ST is already smaller than the full one FT, since we restrict to labels which form an IPO. Still, a lot of observations generated by the IPOs are useless. Actually, if the agent a and the redex R share nothing (i.e. $|a| \cap |R| = \emptyset$) or the redex R does not access to any resources of a, the observation L gives no information about what a can do.

In this section we discuss how, and when, the standard DBTS can be reduced further, but whose bisimilarity coincides with the standard one.

For the rest of the paper we work only with *hard* DBRSs, i.e., DBRSs over $'\mathrm{DBIG}_h(\mathcal{K})$ and $\mathrm{DBIG}_h(\mathcal{K})$ where the place graphs are hard, that is, have no barren roots (see [4, Definition 7.13]).

4.1 Engaged Transition System

A possible way for reducing the transitions in the standard transition system is to consider only transitions where the labels carry some information about the agent. This can be expressed by considering only transitions in whose underlying IPO diagram the redex shares something with the agent, or the label accesses some of agent's resources (i.e. edges).

Definition 15 (engaged transitions). *In* $'\mathrm{DBIG}_h$ *a standard transition of an agent* a *is said to be* engaged *if it can be based on a reaction rule with redex* R *such that* $|a| \cap |R| \neq \emptyset$ *or some nodes of* R *access to resources of* a *(via the IPO-bound). We denote by* ET *the transition system of engaged transitions.*

Notice that this definition extends smoothly the one given by Milner for pure (i.e., output linear) bigraphs [6, Definition 9.10]; in fact, on output-linear bigraphs these definition coincide.

Definition 16 (relative bisimilarity). *A* relative bisimulation *for* ET *(on* ST*) is a symmetric relation* $\mathcal{S}$ *such that if* $a\mathcal{S}b$*, then for every engaged transition* $a \xrightarrow{L}_{\lambda} a'$ *and* Lb *is defined, there exists* b' *such that* $b \xrightarrow{L}_{\lambda} b'$ *in* ST*, and* $a'\mathcal{S}b'$*.*

The relative bisimilarity *for* ET *(on* ST*), denoted by* $\sim_{\mathrm{ST}}^{\mathrm{ET}}$*, is the largest relative bisimulation for* ET *(on* ST*).*

Our aim is to prove that ET is *adequate* for ST, that is, $\sim_{\mathrm{ST}}^{\mathrm{ET}} = \sim_{\mathrm{ST}}$.

To this end, we need to recall and introduce some technical definitions:

Definition 17. *1. A bigraph is* open *if every link is open;*
2. *it is* inner accessible *if every edge and upward outer name is connected to an upward inner name;*
3. *it is* outer accessible *if every edge and downward inner name is connected to a downward outer name;*
4. *it is* accessible *if it is inner and outer accessible;*
5. *it is* inner guarding *if it has no upward inner names and has no site has a root as parent;*

6. *it is* outer guarding *if it has no downward outer names;*
7. *it is* guarding *if it is inner and outer guarding;*
8. *it is* simple *if it has no idle names, no barren roots and no downward inner names, and it is prime, guarding, inner-injective and open.*
9. *it is* pinning *if it has no upward outer names, no barren roots, no two downward outer names are peers, and it is prime, ground and outer accessible.*

Intuitively, a simple bigraph has no edges (i.e., no resources), and the ports of its controls are separately linked to names in the outer interface. Instead, in a pinning bigraph ports are connected only to local edges, each of them is also accessible from exactly one name of the outer interface. Notice that in simple (resp. pinning) bigraphs, the downward (resp. upward) outer interface is empty.

Definition 18. *A DBRS is* simple *if all redexes are simple; it is* pinning *if all redexes are pinning; it is* orthogonal *if all redexes are either simple or pinning.*

Note that all these DBRS have mono and epi redexes.

We recall and give some properties of openness and accessibility.

Proposition 6 (openness properties)

1. *A composition $F \circ G$ is open if and only if both F and G are open.*
2. *Every open bigraph is lean (i.e. has no idle edges).*
3. *If $\boldsymbol{B}$ is an IPO for $\boldsymbol{A}$ and A_1 is open, then B_0 is open.*

Proposition 7 (accessibility properties)

1. *A composition $F \circ G$ is outer (resp. inner) accessible if and only if both F and G are outer (resp. inner) accessible.*
2. *Every inner or outer accessible bigraph is lean.*
3. *If $\boldsymbol{B}$ is an IPO for $\boldsymbol{A}$ and A_1 is outer (resp. inner) accessible, then B_0 is outer (resp. inner) accessible.*

We can now state and prove the main result of the section.

Theorem 1 (adequacy of engaged transitions). *In an orthogonal and linear DBRS equipped with* ST*, the engaged transitions are adequate; that is, relative engaged bisimilarity $\sim^{\mathrm{ET}}_{\mathrm{ST}}$ coincides with $\sim_{\mathrm{ST}}$.*

Proof. The fact that $\sim_{\mathrm{ST}} \subseteq \sim^{\mathrm{ET}}_{\mathrm{ST}}$ is trivial. For the vice versa we shall show that

$$\mathcal{S} = \{(Ca_0, Ca_1) \mid a_0 \sim^{\mathrm{ET}}_{\mathrm{ST}} a_1\} \cup \mathrel{\widehat{=}}$$

is a standard bisimulation up to support equivalence for $\sim_{\mathrm{ST}}$, for details see [6, Proposition 4.5]. This more general result ensures $\sim^{\mathrm{ET}}_{\mathrm{ST}} \subseteq \sim_{\mathrm{ST}}$ by taking $C = id$.

Suppose that $a_0 \sim^{\mathrm{ET}}_{\mathrm{ST}} a_1$. Let $Ca_0 \xrightarrow{M}_{\mu} b'_0$ be a standard transition, with MCa_1 defined. We must find b'_1 such that $Ca_1 \xrightarrow{M}_{\mu} b'_1$ and $(b'_0, b'_1) \in \mathcal{S}^{\widehat{=}}$, where $\mathcal{S}^{\widehat{=}} \triangleq \mathrel{\widehat{=}}\mathcal{S}\mathrel{\widehat{=}}$ is the closure of $\mathcal{S}$ under $\widehat{=}$.

There exists a ground reaction rule (r_0, r'_0) and an IPO (the large square in diagram (a) below) underlying the previous transition of Ca_0. Moreover E_0 is active and $width(E_0)(width(cod(r_0))) = \mu$ and $b'_0 \mathrel{\widehat{=}} E_0 r'_0$. By taking an RPO for (a_0, r_0) relative to (MC, E_0), we get two IPOs as shown in diagram (a).

(a)
$$\begin{tikzcd}
{} \arrow[r, "M"] & {} \\
{} \arrow[u, "C"] \arrow[r, "L"] & {} \arrow[u, "E"'] \\
{} \arrow[u, "a_0"] \arrow[r, "r_0"] & {} \arrow[u, "D_0"'] \arrow[uu, bend right, "E_0"']
\end{tikzcd}$$

(b)
$$\begin{tikzcd}
{} \arrow[r, "L"] & {} \\
{} \arrow[u, "a_1"] \arrow[r, "r_1"] & {} \arrow[u, "D_1"']
\end{tikzcd}$$

(c)
$$\begin{tikzcd}
{} \arrow[r, "M"] & {} \\
{} \arrow[u, "C"] \arrow[r, "L"] & {} \arrow[u, "E"'] \\
{} \arrow[u, "a_1"] \arrow[r, "r_1"] & {} \arrow[u, "D_1"']
\end{tikzcd}$$

Now D_0 is active, so the lower IPO in diagram (a) underlies a transition $a_0 \xrightarrow{L}_\lambda a'_0 \triangleq D_0 r'_0$, where $\lambda = width(D_0)(width(cod(r_0)))$. Also E is active at λ and $b'_0 \hat{=} Ea'_0$. Since MCa_1 is defined we deduce that La_1 is defined; we have to show the existence of a transition $a_1 \xrightarrow{L}_\lambda a'_1$, with underlying IPO as shown in diagram (b). Substituting this IPO for the lower square in diagram (a) we get a transition $Ca_1 \xrightarrow{M}_\mu b'_1 \triangleq Ea'_1$ as shown in diagram (c).

Now, to complete the proof we have to show that $a_1 \xrightarrow{L}_\lambda a'_1$ exists and that $(b'_0, b'_1) \in \mathcal{S}^{\hat{=}}$. This can be done by cases, on how the transition is engaged. $\square$

Notice that in DBTSs, differently from pure bigraphical transition systems, ET restricted to prime agents is not adequate for ST; that is, in general the bisimilarity defined using ET restricted to prime agents does not coincide with $\sim_{\text{ST}}$ on prime agents.

4.2 Definite Engaged Transition System

From the DBTS ET defined in the previous subsection, we want to obtain an abstract DBTS for the corresponding abstract DBRS, obtained by the quotient functor $\mathcal{A} : {}'\mathrm{DBIG}_h(\mathcal{K}) \to \mathrm{DBIG}_h(\mathcal{K})$. To do this, we need to prove that ET is *definite* for ST, that is, that we can infer whether a transition $a \xrightarrow{L}_\lambda a'$ in ST is engaged just by looking at the observation (L, λ) [6, Definition 4.11].

Definition 19 (split, connected). *A split of $F : I \to K$ takes the form:*

$$F = F_1 \circ (F_2 \otimes id_{I_1}) \circ \iota$$

where $F_0 : I_0 \to J$ and $F_1 : J \otimes I_1 \to K$ each have at least one node and $\iota : I \to I_0 \otimes I_1$ is an iso. The split is connected if some port in F_0 is linked to some port in F_1. It is prime if I_0 is prime. F is (prime-)connected *if every (prime) split of F is connected.*

Now we are ready to prove that if the simple redexes of the DBRS satisfies connected condition, then ET is definite.

Now we can prove the main result of this section.

Definition 20. *A label (L, λ) of a transition system is* ambiguous *if it occurs both in an engaged and a not engaged transition. A transition system is* ambiguous *if it has a ambiguous label.*

Proposition 8. *In an orthogonal and linear DBRS, where all the simple redexes are connected, then a label (L, λ) is unambiguous.*

Then by [6, Proposition 4.12] we can conclude that $\sim_{\mathrm{ST}}^{\mathrm{ET}}$ is equal to the absolute one $\sim_{\mathrm{ET}}$ (defined as per Definition 14).

Note that this property applies equally to concrete and abstract DBRSs. Now applying [6, Corollary 4.13] and Theorem 1, we obtain

Proposition 9. *In an orthogonal linear DBRS where all simple redexes are connected:*

1. *The engaged transition system* ET *is definite for* ST.
2. *Absolute engaged bisimilarity $\sim_{\mathrm{ET}}$ coincides with $\sim_{\mathrm{ST}}$.*

We can finally transfer engaged transitions and bisimilarity to the abstract bigraphs. The "engagedness" can be defined only for concrete bigraphs; we say that an abstract transition is *engaged* if it is the image of an engaged transition under $\mathcal{A}$ and we still call engaged bisimilarity the induced bisimilarity under $\mathcal{A}$.

We prove a result for ET similar to Proposition 5 for ST.

Proposition 10. *Let $'\mathcal{D}$ be an orthogonal linear DBRS where all simple redexes are connected, and let $\mathcal{D}$ be its lean-support quotient. Then*

1. *$a \sim_{\mathrm{ET}} b$ if and only if $\mathcal{A}(a)\ \mathcal{A}(\sim_{\mathrm{ET}})\ \mathcal{A}(b)$.*
2. *In $\mathcal{D}$, $\mathcal{A}(\sim_{\mathrm{ET}})$ is a congruence.*

4.3 Extending to Non-hard Abstract Bigraphs

The DBTS above is obtained using the quotient functor $\mathcal{A} : {}'\mathrm{DB{\small IG}}_h(\mathcal{K}) \to \mathrm{DB{\small IG}}_h(\mathcal{K})$, which considers only hard place graphs. As a consequence of this restriction, we cannot use the unit 1 of the sharing products because 1 is not hard. In some cases this is too restrictive; for example, we need to use 1 for encoding the null agent of many process calculi.

A possibility is to introduce a specific zero-arity node $\square$ (called *place node*) which can be used to fill the barren roots; but in this way the structural congruence $\mathbf{0}|P \equiv P$ does not hold; we can only prove that $\mathbf{0}|P \sim P$. In fact, as in [4], we want to quotient the bigraphs by "place equivalence" ($\equiv_\square$), that is, two bigraphs are equal if they differ only by $\square$ nodes.

Let $\Doteq_\square$ be the smallest equivalence including $\Doteq$ and $\equiv_\square$. Then, similarly to $\Doteq$, we obtain the $\Doteq_\square$-quotient functor:

$$\mathcal{A}^\square : {}'\mathrm{DB{\small IG}}_h(\mathcal{K}^\square) \to \mathrm{DB{\small IG}}(\mathcal{K})$$

where $\mathcal{K}^\square$ is the signature $\mathcal{K}$ extended with the place node $\square$.

We want to obtain an abstract (possibly not hard) DBTSs on $\mathcal{K}$ from an hard concrete DBTS on $\mathcal{K}^\square$, using the functor $\mathcal{A}^\square$. As for $\mathcal{A}$, we must prove $\Doteq_\square$ respects ET transitions. We know that $\Doteq$ does so, it remains to show that $\equiv_\square$ respects ET. For this we require that all redexes of the DBRS are *flat*.

Definition 21. *A bigraph is* flat *if no node has a node as parent.*

Now we can prove the same result shown in Proposition 10:

Proposition 11. *Let $'\mathcal{D}(\mathcal{K}^{\Box}, '\mathcal{R})$ be an orthogonal linear hard DBRS (that is, definite and whose redexes are flat), and let $\mathcal{D}(\mathcal{K}, \mathcal{R})$ be its $\Leftrightarrow_{\Box}$-quotient.*

1. *$a \sim_{\mathrm{ET}} b$ if and only if $\mathcal{A}^{\Box}(a)\ \mathcal{A}^{\Box}(\sim_{\mathrm{ET}})\ \mathcal{A}^{\Box}(b)$.*
2. *In $\mathcal{D}$, $\mathcal{A}^{\Box}(\sim_{\mathrm{ET}})$ is a congruence.*

5 An Application: The Fusion Calculus

In this section we apply the theory developed in the previous sections to the Fusion calculus. Recall that the processes of the (monadic) Fusion calculus (without replication) are generated by the following grammar:[2]

$$P, Q ::= \mathbf{0} \mid zx.P \mid \bar{z}x.P \mid P|Q \mid (x)P$$

where the processes are taken up to the structural congruence ($\equiv$), that is the least congruence satisfying the abelian monoid laws for composition and the scope laws and scope extension law:

$$(x)\mathbf{0} \equiv \mathbf{0} \qquad (x)(y)P \equiv (y)(x)P \qquad P|(x)Q \equiv (x)(P|Q) \text{ where } x \notin fn(P).$$

The semantics is defined by the following set of rules (which is a monadic version of that given in [7]), closed under the structural congruence $\equiv$.

$$\mathit{Pref}\ \frac{-}{\alpha.P \xrightarrow{\alpha} P} \qquad \mathit{Par}\ \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \mathit{Open}\ \frac{P \xrightarrow{uz} P',\ u \notin \{z, \bar{z}\}}{(z)P \xrightarrow{(z)uz} P'}$$

$$\mathit{Scope}\ \frac{P \xrightarrow{\{x=y\}} P'}{(y)P \xrightarrow{\mathbf{1}} P'\{x/y\}} \qquad \mathit{Pass}\ \frac{P \xrightarrow{\alpha} P',\ x \notin n(\alpha)}{(x)P \xrightarrow{\alpha} (x)P'} \qquad \mathit{Com}\ \frac{P \xrightarrow{ux} P',\ Q \xrightarrow{\bar{u}y} Q'}{P|Q \xrightarrow{\{x=y\}} P'|Q'}$$

We write (P, φ) to mean that a process has reached the configuration P with associated fusion relation φ. We define $(P, \varphi) \longrightarrow (P', \varphi')$ iff $P \xrightarrow{\psi} P'$ and φ' is the transitive closure of $\varphi \cup \psi$.

Finally, we recall the notions of *fusion bisimilarity* and *hyperequivalence.*

Definition 22. *A* fusion bisimulation *is a symmetric relation $\mathcal{S}$ between processes such that whenever $(P, Q) \in \mathcal{S}$, if $P \xrightarrow{\alpha} P'$ with $bn(\alpha) \cap fn(Q) = \emptyset$, then $Q \xrightarrow{\alpha} Q'$ and $(P'\sigma_\alpha, Q'\sigma_\alpha) \in \mathcal{S}$ (where σ_α denotes the substitutive effect of α).*

P and Q are fusion bisimilar *if $(P, Q) \in \mathcal{S}$ for some fusion bisimulation $\mathcal{S}$.*

A hyperbisimulation *is a substitution closed fusion bisimulation, i.e., an $\mathcal{S}$ such that $(P, Q) \in \mathcal{S}$ implies $(P\sigma, Q\sigma) \in \mathcal{S}$ for any substitution σ. P and Q are* hyperequivalent*, written $P \sim_F Q$, if they are related by a hyperbisimulation.*

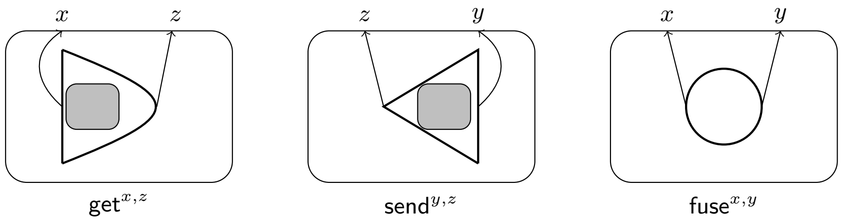


Fig. 1. The controls of the signature for the Fusion calculus

$$\begin{tikzcd}
 & \sum_{[n]\in\{z=y\}} \nabla_n^{[n]} \circ \text{⧗}_n^n \circ \Delta_{[n]}^n \\
{[\![yx.\mathbf{0}|(x)\bar{z}x.\mathbf{0}]\!]_{\{z=y\}}} \arrow[ur] \arrow[dr] & \\
 & {[\![yx.\mathbf{0}|(x)\bar{z}x.\mathbf{0}]\!]_{\{y,z,x\}}}
\end{tikzcd}$$

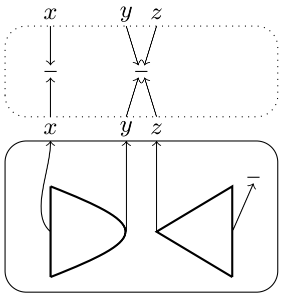


Fig. 2. An example of encoding a fusion process in directed bigraphs

Notice that hyperequivalence only is a congruence, while bisimilarity is not [7].

The signature for representing Fusion processes in directed bigraphs is $\mathcal{K}_F \triangleq \{\mathsf{get}{:}2, \mathsf{send}{:}2, \mathsf{fuse}{:}2\}$, where get, send are passive and fuse is atomic (Figure 1).

A process P is translated to a bigraph of $\mathrm{DBig}(\mathcal{K}_F)$ in two steps, using some algebraic operators of directed bigraphs [1]. First, for X a set of names such that $fn(P) \subseteq X$, we define a bigraph $[\![P]\!]_X : \epsilon \to \langle 1, (\emptyset, X)\rangle$:

$$[\![\mathbf{0}]\!]_X = 1 \curlywedge X \quad [\![P|Q]\!]_X = [\![P]\!]_X \curlywedge [\![Q]\!]_X \quad [\![(x)P]\!]_X = \blacktriangle^x \circ [\![P]\!]_{X \uplus \{x\}}$$
$$[\![zx.P]\!]_X = \mathsf{get}^{x,z} \circ [\![P]\!]_X \quad [\![\bar{z}x.P]\!]_X = \mathsf{send}^{x,z} \circ [\![P]\!]_X \quad \textit{where } x, z \in X$$

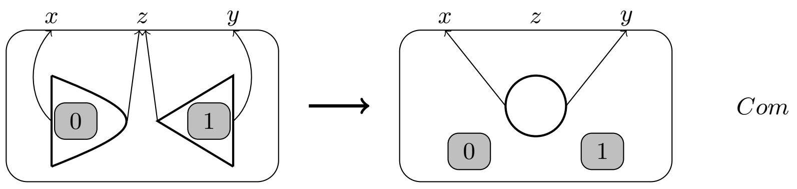
Notice that names in X are represented as outer upward names. In this translation bound names are represented by local (not accessible) edges.

Then, the encoding of a process P under a fusion φ takes the bigraph $[\![P]\!]_{fn(P)}$ and associates to each name in $fn(P)$ an outer accessible edge, according to φ:

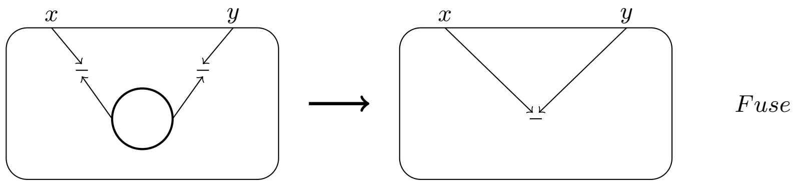
$$[\![P]\!]_\varphi = \left(\sum_{[n]_\varphi \in \varphi} \nabla_n^{[n]_\varphi} \circ \text{⧗}_n^n \circ \Delta_{[n]_\varphi}^n \right) \circ \left([\![P]\!]_{fn(P)} \otimes \sum_{m \in Y \setminus fn(P)} \Delta^m \right)$$

Fusions are represented by linking the fused names (in the outer interface) to the same edge. An example of encoding is given in Figure 2.

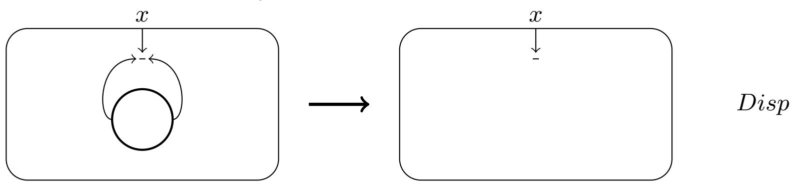
[2] Sum and fusion prefix can be easily encoded in this syntax.



$\mathsf{get}^{x,z} \curlywedge \mathsf{send}^{y,z} \rightarrow \mathsf{fuse}^{x,y} \curlywedge \triangle^{z} \curlywedge id_1 \curlywedge id_1$



$(\text{⧗}^{x}_{x} \otimes \text{⧗}^{y}_{y}) \circ \mathsf{fuse}^{x,y} \rightarrow \triangledown^{x,y}_{z} \circ \blacktriangledown_{z}$



$\text{⧗}^{x}_{x} \circ \triangle^{x}_{y,z} \circ \mathsf{fuse}^{y,z} \rightarrow \blacktriangledown_{x}$

Fig. 3. Reaction rules $\mathcal{R}_F$ for the Fusion calculus

Proposition 12. *Let P and Q be two processes; then $P \equiv Q$ if and only if $[\![P]\!]_\varphi = [\![Q]\!]_\varphi$, for every fusion φ.*

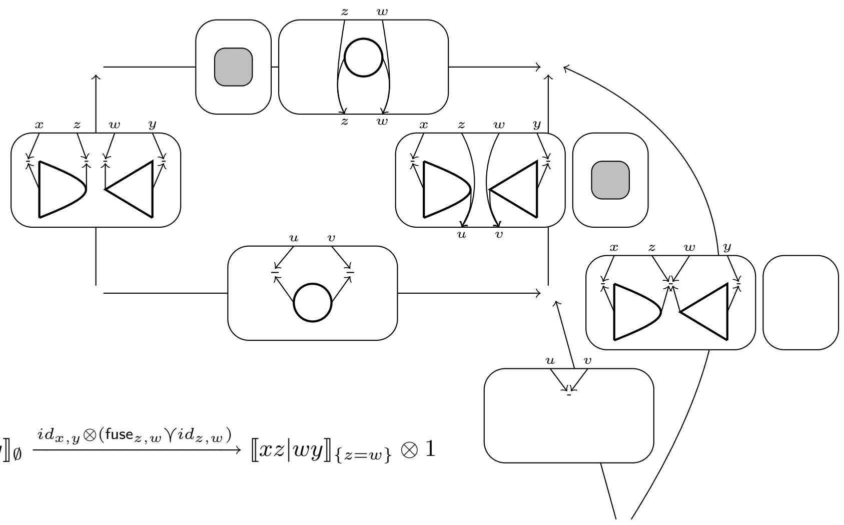
The set of reaction rules ($\mathcal{R}_F$) are shown in Figure 3. Notice that *Com* is simple, instead *Fuse* and *Disp* are pinning; hence this system is orthogonal. Moreover each rule is flat. We denote this DBRS as $\mathcal{D}_F \triangleq \mathcal{D}(\mathcal{K}_F, \mathcal{R}_F)$.

Proposition 13 (Adequacy of the encoding).

1. *if $(P, \varphi) \rightarrow (P', \varphi')$ then $[\![P]\!]_\varphi \longrightarrow^* [\![P']\!]_{\varphi'}$;*
2. *if $[\![P]\!]_\varphi \longrightarrow^* [\![P']\!]_{\varphi'}$ then $(P, \varphi) \rightarrow^* (P', \varphi')$.*

Proof. By induction on the length of the traces. Point 1. is easy. For point 2, first of all note that, by definition of $[\![\cdot]\!]_\varphi$, $[\![P']\!]_{\varphi'}$ has no fuse controls. If $[\![P]\!]_\varphi \longrightarrow^* [\![P']\!]_{\varphi'}$, in the trace there are one or more applications of the *Com* rule in $\mathcal{D}_F$, so we use the *Com* rule of the Fusion on the corresponding P sub-process. We can ignore the *Fuse* and *Disp* rules, because the fusions are performed immediately in the Fusion calculus. $\square$

Working with the abstract bigraphs we obtain the exact match between the Fusion reactions and bigraphic one. Now we want to define the ET bisimilarity for the Fusion calculus. We define ${}'\mathcal{D}_F \triangleq {}'\mathcal{D}(\mathcal{K}^{\square}_F, {}'\mathcal{R}_F)$ to be the concrete DBRS whose precategory of bigraphs is defined on the signature $\mathcal{K}^{\square}_F$, and ${}'\mathcal{R}_F$ are all



$$[\![xz|wy]\!]_{\emptyset} \xrightarrow{id_{x,y}\otimes(\mathsf{fuse}_{z,w}\curlyvee id_{z,w})} [\![xz|wy]\!]_{\{z=w\}} \otimes 1$$

Fig. 4. An example of a non prime engaged transition in $'\mathcal{D}_F$

the reaction rules that are in the preimage of the abstract rules of $\mathcal{D}_F$ via $\mathcal{A}^\square$ (see Figure 3).

First notice that engaged transitions of $'\mathcal{D}_F$ yield a congruential bisimilarity.

Corollary 1. *The bisimilarity $\sim_{\text{ET}}$ is a congruence in $'\mathcal{D}_F$.*

Proof. Since $'\mathcal{R}_F$ is orthogonal and linear, by Theorem 1, ET is adequate for ST in $'\mathcal{D}_F$. Moreover there are no subsumption, then it follows by Proposition 9 that ET is definite for ST and hence $\sim_{\text{ET}}=\sim_{\text{ST}}$. □

Now by Proposition 11, we can derive a congruential bisimilarity in our bigraphical representation of the Fusion.

Corollary 2. *In $\mathcal{D}_F$, the following two sentences are verified:*

1. *$a \sim_{\text{ET}} b$ if and only if $\mathcal{A}^\square(a)$ $\mathcal{A}^\square(\sim_{\text{ET}})$ $\mathcal{A}^\square(b)$;*
2. *$\mathcal{A}^\square(\sim_{\text{ET}})$ is a congruence.*

Clearly, $\sim_{\text{ET}}$ induces a congruence on processes of Fusion calculus. In fact this is the first congruence for Fusion calculus defined only by coinduction (differently from hyperequivalence, which needs a closure under substitutions). However, comparing this congruence with hyperbisimulation or hyperequivalence turns out to be problematic, because the DBTS ET involves also non-prime transitions which are essential to make ET adequate with respect to ST (see Figure 4 for an example of a useful but not prime transition). Thus, when comparing two processes P, Q using $\sim_{\text{ET}}$, we have to consider also non-prime transitions; in these cases, the resulting agents are non-prime, and their connection with the descendants of P and Q in the original semantics is still unclear.

6 Conclusions

In this paper, we have presented *directed bigraphical reactive systems* and *directed bigraphical transition systems*, that is wide reactive and transition systems built over directed bigraphs. We have shown that the bisimilarity induced by the IPO construction is always a congruence; moreover, under a mild condition, this bisimilarity can be characterized by a smaller LTS whose transitions (called "engaged") are only those really relevant for the agents. As an application, we have presented the first encoding of the Fusion calculus as a DBRS; then, using the general constructions given in this paper, we have defined a bisimilarity for Fusion which is also a congruence, without the need of a closure under substitutions.

The exact relation between this equivalence and those defined in the literature (i.e., hyperbisimilarity and hyperequivalence) is still under investigation. The issue is that for DBRSs, engaged transitions of prime agents may include also non-prime transitions, yielding non-prime agents as results. This happens in the case of Fusion calculus, and these transitions are not easily interpreted in terms of the labelled transition systems by which hyperbisimilarity is defined.

Another possible future work concerns the application of the theory developed in this paper for verification of properties of systems represented as DBRSs. Beside using standard bisimilarity for checking behavioural equivalence, the compact DBTS can be used also for model checking purposes.

Acknowledgements. The authors wish to thank Robin Milner and the anonymous referees for their helpful comments.

References

1. Grohmann, D., Miculan, M.: An algebra for directed bigraphs. In: Mackie, I., Plump, D. (eds.) Pre-proceedings of TERMGRAPH 2007. ENTCS, Elsevier, Amsterdam (2007)
2. Grohmann, D., Miculan, M.: Directed bigraphs. In: Proc. XXIII MFPS. ENTCS, vol. 173, pp. 121–137. Elsevier, Amsterdam (2007)
3. Jensen, O.H., Milner, R.: Bigraphs and transitions. In: Proc. POPL (2003)
4. Jensen, O.H., Milner, R.: Bigraphs and mobile processes (revised). Technical report UCAM-CL-TR-580, Computer Laboratory, University of Cambridge (2004)
5. Milner, R.: Bigraphical reactive systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 16–35. Springer, Heidelberg (2001)
6. Milner, R.: Pure bigraphs: Structure and dynamics. Inf. Comput. 204(1) (2006)
7. Parrow, J., Victor, B.: The fusion calculus: Expressiveness and symmetry in mobile processes. In: Proceedings of LICS '98, pp. 176–185. IEEE Computer Society Press, Los Alamitos (1998)
8. Sassone, V., Sobociński, P.: Reactive systems over cospans. In: Proc. LICS 2005, 20th IEEE Symposium on Logic in Computer Science, Chicago, IL, USA, 26-29 June 2005, pp. 311–320. IEEE Computer Society, Los Alamitos (2005)

Asynchronous Games: Innocence Without Alternation[⋆]

Paul-André Melliès and Samuel Mimram

Équipe PPS, CNRS and Université Paris 7, 2 place Jussieu, case 7017, 75251 Paris cedex 05, France
mellies@pps.jussieu.fr, smimram@pps.jussieu.fr

Abstract. The notion of innocent strategy was introduced by Hyland and Ong in order to capture the interactive behaviour of λ-terms and PCF programs. An innocent strategy is defined as an alternating strategy with partial memory, in which the strategy plays according to its view. Extending the definition to non-alternating strategies is problematic, because the traditional definition of views is based on the hypothesis that Opponent and Proponent alternate during the interaction. Here, we take advantage of the diagrammatic reformulation of alternating innocence in asynchronous games, in order to provide a tentative definition of innocence in non-alternating games. The task is interesting, and far from easy. It requires the combination of true concurrency and game semantics in a clean and organic way, clarifying the relationship between asynchronous games and concurrent games in the sense of Abramsky and Melliès. It also requires an interactive reformulation of the usual acyclicity criterion of linear logic, as well as a directed variant, as a scheduling criterion.

1 Introduction

The alternating origins of game semantics. Game semantics was invented (or reinvented) at the beginning of the 1990s in order to describe the dynamics of proofs and programs. It proceeds according to the principles of trace semantics in concurrency theory: every program and proof is interpreted by the sequences of interactions, called *plays*, that it can have with its environment. The novelty of game semantics is that this set of plays defines a *strategy* which reflects the interactive behaviour of the program inside the *game* specified by the type of the program.

Game semantics was originally influenced by a pioneering work by Joyal [16] building a category of games (called Conway games) and alternating strategies. In this setting, a game is defined as a decision tree (or more precisely, a dag) in which every edge, called *move*, has a polarity indicating whether it is played by the program, called Proponent, or by the environment, called Opponent. A play is alternating when Proponent and Opponent alternate strictly – that is, when neither of them plays two moves in a row. A strategy is alternating when it contains only alternating plays.

The category of alternating strategies introduced by Joyal was later refined by Abramsky and Jagadeesan [2] in order to characterize the dynamic behaviour of proofs

⋆ This work has been supported by the ANR Invariants algébriques des systèmes informatiques (INVAL).

in (multiplicative) linear logic. The key idea is that the tensor product of linear logic, noted $\otimes$, may be distinguished from its dual, noted Γ, by enforcing a *switching policy* on plays – ensuring for instance that a strategy of $A \otimes B$ reacts to an Opponent move played in the subgame A by playing a Proponent move in the same subgame A.

The notion of *pointer game* was then introduced by Hyland and Ong, and independently by Nickau, in order to characterize the dynamic behaviour of programs in the programming language PCF – a simply-typed λ-calculus extended with recursion, conditional branching and arithmetical constants. The programs of PCF are characterized dynamically as particular kinds of strategies with partial memory – called *innocent* because they react to Opponent moves according to their own *view* of the play. This view is itself a play, extracted from the current play by removing all its "invisible" or "inessential" moves. This extraction is performed by induction on the length of the play, using the pointer structure of the play, and the hypothesis that Proponent and Opponent alternate strictly.

This seminal work on pointer games led to the first generation of game semantics for programming languages. The research programme – mainly guided by Abramsky and his collaborators – was extraordinarily successful: by relaxing the innocence constraint on strategies, it suddenly became possible to characterize the interactive behaviour of programs written in PCF (or in a call-by-value variant) extended with imperative features like states, references, etc. However, because Proponent and Opponent strictly alternate in the original definition of pointer games, these game semantics focus on sequential languages like Algol or ML, rather than on concurrent languages.

Concurrent games. This convinced a little community of researchers to work on the foundations of non-alternating games – where Proponent and Opponent are thus allowed to play several moves in a row at any point of the interaction. Abramsky and Melliès [3] introduced a new kind of game semantics to that purpose, based on *concurrent games* – see also [1]. In that setting, games are defined as partial orders (or more precisely, complete lattices) of *positions*, and strategies as *closure operators* on these partial orders. Recall that a closure operator σ on a partial order D is a function $\sigma : D \longrightarrow D$ satisfying the following properties:

$$\begin{array}{llll}
(1) & \sigma \text{ is increasing:} & \forall x \in D, & x \leq \sigma(x), \\
(2) & \sigma \text{ is idempotent:} & \forall x \in D, & \sigma(x) = \sigma(\sigma(x)), \\
(3) & \sigma \text{ is monotone:} & \forall x, y \in D, & x \leq y \Rightarrow \sigma(x) \leq \sigma(y).
\end{array}$$

The order on positions $x \leq y$ reflects the intuition that the position y contains more information than the position x. Typically, one should think of a position x as a set of moves in a game, and $x \leq y$ as set inclusion $x \subseteq y$. Now, Property (1) expresses that a strategy σ which transports the position x to the position $\sigma(x)$ increases the amount of information. Property (2) reflects the intuition that the strategy σ delivers all its information when it transports the position x to the position $\sigma(x)$, and thus transports the position $\sigma(x)$ to itself. Property (3) is both fundamental and intuitively right, but also more subtle to justify. Note that the interaction induced by such a strategy σ is possibly non-alternating, since the strategy transports the position x to the position $\sigma(x)$ by "playing in one go" all the moves appearing in $\sigma(x)$ but not in x.

Asynchronous transition systems. Every closure operator σ is characterized by the set fix(σ) of its fixpoints, that is, the positions x satisfying $x = \sigma(x)$. So, a strategy is expressed alternatively as a set of positions (the set of fixpoints of the closure operator) in concurrent games, and as a set of alternating plays in pointer games. In order to understand how the two formulations of strategies are related, one should start from an obvious analogy with concurrency theory: pointer games define an *interleaving semantics* (based on sequences of transitions) whereas concurrent games define a *truly concurrent semantics* (based on sets of positions, or states) of proofs and programs. Now, Mazurkiewicz taught us this important lesson: a truly concurrent semantics may be regarded as an interleaving semantics (typically a transition system) equipped with *asynchronous tiles* – represented diagrammatically as 2-dimensional tiles

$$\begin{tikzcd} & x \arrow[dl, "m"'] \arrow[dr, "n"] & \\ y_1 \arrow[dr, "n"'] & \sim & y_2 \arrow[dl, "m"] \\ & z & \end{tikzcd} \qquad (1)$$

expressing that the two transitions m and n from the state x are *independent*, and consequently, that their scheduling does not matter from a truly concurrent point of view. This additional structure induces an equivalence relation on transition paths, called *homotopy*, defined as the smallest congruence relation $\sim$ identifying the two schedulings $m \cdot n$ and $n \cdot m$ for every tile of the form (1). The word *homotopy* should be understood mathematically as (directed) homotopy in the topological presentation of asynchronous transition systems as *n-cubical sets* [15]. This 2-dimensional refinement of usual 1-dimensional transition systems enables to express simultaneously the interleaving semantics of a program as the set of transition paths it generates, and its truly concurrent semantics, as the homotopy classes of these transition paths. When the underlying 2-dimensional transition system is contractible in a suitable sense, explained later, these homotopy classes coincide in fact with the positions of the transition system.

Asynchronous games. Guided by these intuitions, Melliès introduced the notion of *asynchronous game*, which unifies in a surprisingly conceptual way the two heterogeneous notions of pointer game and concurrent game. Asynchronous games are played on asynchronous (2-dimensional) transition systems, where every transition (or move) is equipped with a *polarity*, expressing whether it is played by Proponent or by Opponent. A *play* is defined as a path starting from the root (noted $*$) of the game, and a *strategy* is defined as a well-behaved set of alternating plays, in accordance with the familiar principles of pointer games. Now, the difficulty is to understand how (and when) a strategy defined as a set of plays may be reformulated as a set of positions, in the spirit of concurrent games.

The first step in the inquiry is to observe that the asynchronous tiles (1) offer an alternative way to describe *justification pointers* between moves. For illustration, consider the boolean game $\mathbb{B}$, where Opponent starts by asking a question q, and Proponent answers by playing either `true` or `false`. The game is represented by the decision tree

$$\begin{tikzcd} & * \arrow[d, "\mathtt{q}"] & \\ & q \arrow[dl, "\mathtt{false}"'] \arrow[dr, "\mathtt{true}"] & \\ F & & V \end{tikzcd} \qquad (2)$$

where $*$ is the root of the game, and the three remaining positions are called q, F and V (V for "Vrai" in French). At this point, since there is no concurrency involved, the game may be seen either as an asynchronous game, or as a pointer game. Now, the game $\mathbb{B} \otimes \mathbb{B}$ is constructed by taking two boolean games "in parallel." It simulates a very simple computation device, containing two boolean memory cells. In a typical interaction, Opponent starts by asking with $\mathtt{q}_L$ the value of the left memory cell, and Proponent answers $\mathtt{true}_L$; then, Opponent asks with $\mathtt{q}_R$ the value of the right memory cell, and Proponent answers $\mathtt{false}_R$. The play is represented as follows in pointer games:

$$\begin{tikzcd}
\mathtt{q}_L & \cdot & \mathtt{true}_L & \cdot & \mathtt{q}_R & \cdot & \mathtt{false}_R
\arrow[from=1-3, to=1-1, bend right]
\arrow[from=1-7, to=1-5, bend right]
\end{tikzcd}$$

The play contains two justification pointers, each one represented by an arrow starting from a move and leading to a previous move. Typically, the justification pointer from the move $\mathtt{true}_L$ to the move $\mathtt{q}_L$ indicates that the answer $\mathtt{true}_L$ is necessarily played after the question $\mathtt{q}_L$. The same situation is described using 2-dimensional tiles in the asynchronous game $\mathbb{B} \otimes \mathbb{B}$ below:

$$\begin{tikzcd}
 & & * \otimes * \arrow[dl, "\mathtt{q}_L"'] \arrow[dr, "\mathtt{q}_R"] & & \\
 & q \otimes * \arrow[dl, "\mathtt{true}_L"'] \arrow[dr, "\mathtt{q}_R"] & \sim & * \otimes q \arrow[dl, "\mathtt{q}_L"'] \arrow[dr, "\mathtt{false}_R"] & \\
V \otimes * \arrow[dr, "\mathtt{q}_R"'] & \sim & q \otimes q \arrow[dl, "\mathtt{true}_L"'] \arrow[dr, "\mathtt{false}_R"] & \sim & * \otimes F \arrow[dl, "\mathtt{q}_L"] \\
 & V \otimes q \arrow[dr, "\mathtt{false}_R"'] & \sim & q \otimes F \arrow[dl, "\mathtt{true}_L"] & \\
 & & V \otimes F & &
\end{tikzcd} \qquad (3)$$

The justification pointer between the answer $\mathtt{true}_L$ and its question $\mathtt{q}_L$ is replaced here by a *dependency* relation between the two moves, ensuring that the move $\mathtt{true}_L$ cannot be permuted before the move $\mathtt{q}_L$. The dependency itself is expressed by a "topological" obstruction: the lack of a 2-dimensional tile permuting the transition $\mathtt{true}_L$ before the transition $\mathtt{q}_L$ in the asynchronous game $\mathbb{B} \otimes \mathbb{B}$.

This basic correspondence between justification pointers and asynchronous tiles allows a reformulation of the original definition of *innocent strategy* in pointer games (based on views) in the language of asynchronous games. Surprisingly, the reformulation leads to a purely local and diagrammatic definition of innocence in asynchronous games, which does not mention the notion of view any more. This diagrammatic reformulation leads then to the important discovery that innocent strategies are *positional* in the following sense. Suppose that two alternating plays $s, t : * \longrightarrow x$ with the same target position x are elements of an innocent strategy σ, and that m is an Opponent move from position x. Suppose moreover that the two plays s and t are equivalent modulo homotopy. Then, the innocent strategy σ extends the play $s \cdot m$ with a Proponent move n if and only if it extends the play $t \cdot m$ with the same Proponent move n. Formally:

$$s \cdot m \cdot n \in \sigma \quad \text{and} \quad s \sim t \quad \text{and} \quad t \in \sigma \quad \text{implies} \quad t \cdot m \cdot n \in \sigma. \qquad (4)$$

From this follows that every innocent strategy σ is characterized by the set of *positions* (understood here as homotopy classes of plays) reached in the asynchronous game. This set of positions defines a closure operator, and thus a strategy in the sense of concurrent games. Asynchronous games offer in this way an elegant and unifying point of view on pointer games and concurrent games.

Concurrency in game semantics. There is little doubt that a new generation of game semantics is currently emerging along this foundational work on concurrent games. We see at least three converging lines of research. First, authors trained in game semantics – Ghica, Laird and Murawski – were able to characterize the interactive behaviour of various concurrent programming languages like Parallel Algol [12] or an asynchronous variant of the π-calculus [17] using directly (and possibly too directly) the language of pointer games. Then, authors trained in proof theory and game semantics – Curien and Faggian – relaxed the sequentiality constraints required by Girard on *designs* in ludics, leading to the notion of L-net [9] which lives at the junction of syntax (expressed as proof nets) and game semantics (played on event structures). Finally, and more recently, authors trained in process calculi, true concurrency and game semantics – Varacca and Yoshida – were able to extend Winskel's truly concurrent semantics of CCS, based on event structures, to a significant fragment of the π-calculus, uncovering along the way a series of nice conceptual properties of *confusion-free* event structures [25].

So, a new generation of game semantics for concurrent programming languages is currently emerging... but their various computational models are still poorly connected. We would like a regulating theory here, playing the role of Hyland and Ong pointer games in traditional (that is, alternating) game semantics. Asynchronous games are certainly a good candidate, because they combine interleaving semantics and causal semantics in a harmonious way. Unfortunately, they were limited until now to alternating strategies [20]. The key contribution of this paper is thus to extend the asynchronous framework to non-alternating strategies in a smooth way.

Asynchronous games without alternation. One particularly simple recipe to construct an asynchronous game is to start from a *partial order* of events where, in addition, every event has a polarity, indicating whether it is played by Proponent or Opponent. This partial order $(M, \preceq)$ is then equipped with a *compatibility relation* satisfying a series of suitable properties – defining what Winskel calls an *event structure*. A *position* x of the asynchronous game is defined as a set of *compatible* events (or moves) closed under the "causality" order:

$$\forall m, n \in M, \qquad m \preceq n \quad \text{and} \quad n \in x \quad \text{implies} \quad m \in x.$$

Typically, the boolean game $\mathbb{B}$ described in (2) is generated by the event structure

$$\begin{tikzcd} & \mathtt{q} \arrow[dl, dash] \arrow[dr, dash] & \\ \mathtt{true} & & \mathtt{false} \end{tikzcd}$$

where $\mathtt{q}$ is an Opponent move, and $\mathtt{false}$ and $\mathtt{true}$ are two *incompatible* Proponent moves, with the positions q, V, F defined as $q = \{\mathtt{q}\}$, $V = \{\mathtt{q}, \mathtt{true}\}$ and $F = \{\mathtt{q}, \mathtt{false}\}$. The tensor product $\mathbb{B} \otimes \mathbb{B}$ of two boolean games is then generated by putting side by side the two event structures, in the expected way. The resulting asynchronous game looks like a flower with four petals, one of them described in (3). More generally, every formula of linear logic defines an event structure – which generates in turn the asynchronous game associated to the formula. For instance, the event structure induced by the formula

$$(\mathbb{B} \otimes \mathbb{B}) \multimap \mathbb{B} \tag{5}$$

contains the following partial order of compatible events:

$$\begin{tikzcd}
 & \mathtt{q} \arrow[dl, dash] \arrow[dd, dash] \arrow[dr, dash] & \\
\mathtt{q}_L \arrow[d, dash] & & \mathtt{q}_R \arrow[d, dash] \\
\mathtt{true}_L & \mathtt{false} & \mathtt{false}_R
\end{tikzcd} \tag{6}$$

which may be seen alternatively as a (maximal) position in the asynchronous game associated to the formula.

This game implements the interaction between a boolean function (Proponent) of type (5) and its two arguments (Opponent). In a typical play, Opponent starts by playing the move $\mathtt{q}$ asking the value of the boolean output; Proponent reacts by asking with $\mathtt{q}_L$ the value of the left input, and Opponent answers $\mathtt{true}_L$; then, Proponent asks with $\mathtt{q}_R$ the value of the right input, and Opponent answers $\mathtt{false}_R$; at this point only, using the knowledge of its two arguments, Proponent answers $\mathtt{false}$ to the initial question:

$$\mathtt{q} \cdot \mathtt{q}_L \cdot \mathtt{true}_L \cdot \mathtt{q}_R \cdot \mathtt{false}_R \cdot \mathtt{false} \tag{7}$$

Of course, Proponent could have explored its two arguments in the other order, from right to left, this inducing the play

$$\mathtt{q} \cdot \mathtt{q}_R \cdot \mathtt{false}_R \cdot \mathtt{q}_L \cdot \mathtt{true}_L \cdot \mathtt{false} \tag{8}$$

The two plays start from the empty position $*$ and reach the same position of the asynchronous game. They may be seen as different linearizations (in the sense of order theory) of the partial order (6) provided by the game. Each of these linearizations may be represented by adding causality (dotted) edges between moves to the original partial order (6), in the following way:

$$\begin{tikzcd}
 & \mathtt{q} \arrow[dl, dash] \arrow[ddddd, dash] \arrow[dddr, dash] & & & & \mathtt{q} \arrow[dddl, dash] \arrow[ddddd, dash] \arrow[dr, dash] & \\
\mathtt{q}_L \arrow[d, dash] & & & & & & \mathtt{q}_R \arrow[d, dash] \\
\mathtt{true}_L \arrow[drr, dotted, dash] & & & & & & \mathtt{true}_R \arrow[dllll, dotted, dash] \\
 & & \mathtt{q}_R \arrow[d, dash] & & \mathtt{q}_L \arrow[d, dash] & & \\
 & & \mathtt{false}_R \arrow[dl, dotted, dash] & & \mathtt{false}_L \arrow[dr, dotted, dash] & & \\
 & \mathtt{false} & & & & \mathtt{false} &
\end{tikzcd} \tag{9}$$

The play (7) is an element of the strategy representing the *left* implementation of the *strict conjunction*, whereas the play (8) is an element of the strategy representing its *right* implementation. Both of these strategies are alternating. Now, there is also a *parallel* implementation, where the conjunction asks the value of its two arguments at the same time. The associated strategy is not alternating anymore: it contains the play (7) and the play (8), and moreover, all the (possibly non-alternating) linearizations of the following partial order:

$$\begin{tikzcd}
 & \mathtt{q} \arrow[dl, dash] \arrow[dr, dash] \arrow[ddd, dash] & \\
\mathtt{q}_L \arrow[d, dash] & & \mathtt{q}_R \arrow[d, dash] \\
\mathtt{true}_L \arrow[dr, dotted, dash] & & \mathtt{false}_R \arrow[dl, dotted, dash] \\
 & \mathtt{false} &
\end{tikzcd} \tag{10}$$

This illustrates an interesting phenomenon, of a purely concurrent nature: every play s of a concurrent strategy σ coexists with other plays t in the strategy, having the same target position x – and in fact, equivalent modulo homotopy. It is possible to reconstruct from this set of plays a *partial order* on the events of x, refining the partial order on events provided by the game. This partial order describes entirely the strategy σ under the position x: more precisely, the set of plays in σ reaching the position x coincides with the set of the linearizations of the partial order.

Our definition of innocent strategy will ensure the existence of such an underlying "causality order" for every position x reached by the strategy. Every innocent strategy will then define an event structure, obtained by putting together all the induced partial orders. The construction requires refined tools from the theory of asynchronous transition systems, and in particular the fundamental notion of *cube property*.

Ingenuous strategies. We introduce in Section 4 the notion of *ingenuous* strategy, defined as a strategy regulated by an underlying "causality order" on moves for every reached position, and satisfying a series of suitable diagrammatic properties. One difficulty is that ingenuous strategies do not compose properly. Consider for instance the ingenuous strategy σ of type $\mathbb{B} \otimes \mathbb{B}$ generated by the partial order:

$$\begin{tikzcd}
\mathtt{q}_L \arrow[d, dash] \arrow[dr, dotted, dash] & \mathtt{q}_R \arrow[d, dash] \\
\mathtt{true}_L & \mathtt{false}_R
\end{tikzcd} \tag{11}$$

The strategy answers $\mathtt{true}_L$ to the question $\mathtt{q}_L$, but answers $\mathtt{false}_R$ to the question $\mathtt{q}_R$ only if the question $\mathtt{q}_L$ has been already asked. Composing the strategy σ with the *right* implementation of the strict conjunction pictured on the right-hand side of (9) induces a play $\mathtt{q} \cdot \mathtt{q}_R$ stopped by a *deadlock* at the position $\{\mathtt{q}, \mathtt{q}_R\}$. On the other hand, composing the strategy with the *left* or the *parallel* implementation is fine, and leads to a complete interaction.

This dynamic phenomenon is better understood by introducing two new binary connectives $\oslash$ and $\ominus$ called "before" and "after", describing sequential composition in asynchronous games. The game $A \oslash B$ is defined as the 2-dimensional restriction of the game $A \otimes B$ to the plays s such that every move played before a move in A is also in A; or equivalently, every move played after a move B is also in B. The game $A \ominus B$ is simply defined as the game $B \oslash A$, where the component B thus starts.

The ingenuous strategy σ in $\mathbb{B} \otimes \mathbb{B}$ specializes to a strategy in the subgame $\mathbb{B} \oslash \mathbb{B}$, which reflects it, in the sense that every play $s \in \sigma$ is equivalent modulo homotopy to a play $t \in \sigma$ in the subgame $\mathbb{B} \oslash \mathbb{B}$. This is not true anymore when one specializes the strategy σ to the subgame $\mathbb{B} \ominus \mathbb{B}$, because the play $\mathtt{q}_L \cdot \mathtt{true}_L \cdot \mathtt{q}_R \cdot \mathtt{false}_R$ is an element of σ which is not equivalent modulo homotopy to any play $t \in \sigma$ in the subgame $\mathbb{B} \ogreaterthan \mathbb{B}$. For that reason, we declare that the strategy σ is innocent in the game $\mathbb{B} \olessthan \mathbb{B}$ but *not* in the game $\mathbb{B} \otimes \mathbb{B}$.

Innocent strategies. This leads to an interactive criterion which tests dynamically whether an ingenuous strategy σ is *innocent* for a given formula of linear logic. The criterion is based on *scheduling conditions* which recast, in the framework of non-alternating games, the *switching conditions* formulated by Abramsky and Jagadeesan for alternating games [2]. The idea is to *switch* every tensor product $\otimes$ of the formula as $\olessthan$ or $\ogreaterthan$ and to test whether every play s in the strategy σ is equivalent modulo homotopy to a play $t \in \sigma$ in the induced subgame. Every such switching reflects a choice of scheduling by the counter-strategy: an innocent strategy is thus a strategy flexible enough to adapt to *every* scheduling of the tensor products by Opponent. An ingenuous strategy satisfies the scheduling criterion if and only if the underlying proof-structure satisfies a directed (and more liberal) variant of the acyclicity criterion introduced by Girard [13] and reformulated by Danos and Regnier [10]. A refinement based on the notion of synchronized clusters of moves enables then to strengthen the scheduling criterion, and to make it coincide with the usual non-directed acyclicity criterion.

We will establish in Section 4 that every ingenuous strategy may be seen alternatively as a closure operator, whose fixpoints are precisely the *halting positions* of the strategy. This connects (non alternating) asynchronous games to concurrent games. However, there is a subtle mismatch between the interaction of two ingenuous strategies seen as sets of plays, and seen as sets of positions. Typically, the right implementation of the strict conjunction in (9) composed to the strategy σ in (11) induces two different fixpoints in the concurrent game model: the deadlock position $\{\mathtt{q}, \mathtt{q}_R\}$ reached during the asynchronous interaction, and the complete position $\{\mathtt{q}, \mathtt{q}_L, \mathtt{true}_L, \mathtt{q}_R, \mathtt{true}_R, \mathtt{false}\}$ which is never reached interactively. The innocence assumption is precisely what ensures that this will never occur: the fixpoint computed in the concurrent game model is unique, and coincides with the position eventually reached in the asynchronous game model. In particular, innocent strategies compose properly.

2 The Cube Property

The *cube property* expresses a fundamental causality principle in the diagrammatic language of asynchronous transition systems [5,24,26]. The property is related to stability in the sense of Berry [6]. It was first noticed by Nielsen, Plotkin and Winskel in [21], then reappeared in [22] and [14,18] and was studied thoroughly by Kuske in his PhD thesis; see [11] for a survey. The most natural way to express the property is to start from what we call an *asynchronous graph*. Recall that a *graph* $G = (V, E, \partial_0, \partial_1)$ consists of a set V of vertices (or *positions*), a set E of edges (or *transitions*), and two functions $\partial_0, \partial_1 : E \to V$ called respectively source and target functions. An *asynchronous graph* $G = (G, \diamond)$ is a graph G together with a relation $\diamond$ on coinitial and cofinal transition paths of length 2. Every relation $s \diamond t$ is represented diagrammatically as a 2-dimensional tile

$$\begin{tikzcd}
 & x \arrow[dl, "m"'] \arrow[dr, "n"] & \\
y_1 \arrow[dr, "p"'] & \sim & y_2 \arrow[dl, "q"] \\
 & z &
\end{tikzcd} \tag{12}$$

where $s = m \cdot p$ and $t = n \cdot q$. In this diagram, the transition q is intuitively the *residual* of the transition m after the transition n. One requires the two following properties for every asynchronous tile:

1. $m \neq n$ and $p \neq q$,
2. the pair of transitions (n, q) is uniquely determined by the pair of transitions (m, p), and conversely.

The main difference with the asynchronous tile (1) occurring in the asynchronous transition systems defined in [26,23] is that the transitions are not labelled by events: so, the 2-dimensional structure is purely "geometric" and not deduced from an independence relation on events. What matters is that the 2-dimensional structure enables one to define a homotopy relation $\sim$ on paths in exactly the same way. Moreover, every homotopy class of a path $s = m_1 \cdots m_k$ coincides with the set of linearizations of a partial order on its transitions if, and only if, the asynchronous graph satisfies the following *cube property*:

> *Cube property:* a hexagonal diagram induced by two coinitial and cofinal paths $m \cdot n \cdot o : x \longrightarrow\!\!\!\!\!\rightarrow y$ and $p \cdot q \cdot r : x \longrightarrow\!\!\!\!\!\rightarrow y$ is filled by 2-dimensional tiles as pictured in the left-hand side of the diagram below, if and only it is filled by 2-dimensional tiles as pictured in the right-hand side of the diagram:

$$\begin{tikzcd}
 & x \arrow[rr, "p"] \arrow[dl, "m"'] \arrow[dd] \arrow[ddrr, phantom, "\sim"] & & x_2 \arrow[dd, "q"] \\
x_1 \arrow[dd, "n"'] \arrow[dr, phantom, "\sim"] & & & \\
 & x_3 \arrow[rr] \arrow[dl] \arrow[dr, phantom, "\sim"] & & y_1 \arrow[dl, "r"] \\
y_2 \arrow[rr, "o"'] & & y &
\end{tikzcd}
\quad \Longleftrightarrow \quad
\begin{tikzcd}
 & x \arrow[rr, "p"] \arrow[dl, "m"'] \arrow[dr, phantom, "\sim"] & & x_2 \arrow[dl] \arrow[dd, "q"] \\
x_1 \arrow[rr] \arrow[dd, "n"'] \arrow[ddrr, phantom, "\sim"] & & y_3 \arrow[dd] \arrow[dr, phantom, "\sim"] & \\
 & & & y_1 \arrow[dl, "r"] \\
y_2 \arrow[rr, "o"'] & & y &
\end{tikzcd}$$

The cube property is for instance satisfied by every asynchronous transition system and every transition system with independence in the sense of [26,23]. The correspondence between homotopy classes and sets of linearizations of a partial order adapts, in our setting, a standard result on pomsets and asynchronous transition systems with dynamic independence due to Bracho, Droste and Kuske [7].

Every asynchronous graph G equipped with a distinguished initial position (noted $*$) induces an asynchronous graph $[G]$ whose positions are the homotopy classes of paths starting from the position $*$, and whose edges $m : [s] \longrightarrow [t]$ between the homotopy classes of the paths $s : * \longrightarrow\!\!\!\!\!\rightarrow x$ and $t : * \longrightarrow\!\!\!\!\!\rightarrow y$ are the edges $m : x \longrightarrow y$ such that $s \cdot m \sim t$. When the original asynchronous graph G satisfies the cube property, the resulting asynchronous graph $[G]$ is "contractible" in the sense that every two coinitial and cofinal paths are equivalent modulo homotopy.

So, we will suppose from now on that all our asynchronous graphs satisfy the cube property and are therefore contractible. The resulting framework is very similar to the domain of configurations of an event structure. Indeed, every contractible asynchronous graph defines a partial order on its set of positions, defined by reachability: $x \leq y$ when $x \twoheadrightarrow y$. Moreover, this order specializes to a finite distributive lattice under every position x, rephrasing – by Birkhoff representation theorem – the fact already mentioned that the homotopy class of a path $* \twoheadrightarrow x$ coincides with the linearizations of a partial order on its transitions. Finally, every transition may be labelled by an "event" representing the transition modulo a "zig-zag" relation, identifying the moves m and q in every asynchronous tile (12). The idea of "zig-zag" is folklore: it appears for instance in [23] in order to translate a transition system with independence into a labelled event structure.

3 Positionality in Asynchronous Games

Before considering 2-Player games, we express the notion of a positional strategy in 1-Player games. A 1-Player game $(G, *)$ is simply defined as an asynchronous graph G together with a distinguished initial position $*$. A play is defined as a path starting from $*$, and a strategy is defined as a set of plays of the 1-Player game, closed under prefix. A strategy σ is called *positional* when for every three paths $s, t : * \twoheadrightarrow x$ and $u : x \twoheadrightarrow y$, we have

$$s \cdot u \in \sigma \quad \text{and} \quad s \sim t \quad \text{and} \quad t \in \sigma \quad \text{implies} \quad t \cdot u \in \sigma. \tag{13}$$

This adapts the definition of (4) to a non-polarized setting and extends to the non-alternating setting. Note that a positional strategy is the same thing as a subgraph of the 1-Player game, where every position is reachable from $*$ inside the subgraph. This subgraph inherits a 2-dimensional structure from the underlying 1-Player game; it thus defines an asynchronous graph, denoted G_σ.

The advantage of considering asynchronous graphs instead of event structures appears at this point: the "event" associated to a transition is deduced from the 2-dimensional geometry of the graph. So, the "event" associated to a transition m in the graph G_σ describes the "causality cascade" leading the strategy to play the transition m ; whereas the "event" associated to the same transition m in the 1-Player game G is simply the move of the game. This subtle difference is precisely what underlies the distinction between the formula (5) and the various strategies (9) and (10). For instance, there are three "events" associated to the output move `false` in the parallel implementation of the strict conjunction, each one corresponding to a particular pair of inputs $(\texttt{true}, \texttt{false})$, $(\texttt{false}, \texttt{false})$, and $(\texttt{false}, \texttt{true})$. This phenomenon is an avatar of Berry stability, already noticed in [19].

From now on, we only consider positional strategies satisfying two additional properties:

1. forward compatibility preservation: every asynchronous tile of the shape (12) in the 1-Player game G belongs to the subgraph G_σ of the strategy σ when its two coinitial

transitions $m : x \longrightarrow y_1$ and $n : x \longrightarrow y_2$ are transitions in the subgraph G_σ. Diagrammatically,

$$\begin{tikzcd}[column sep=small] & x \arrow[dl, "\sigma\ni m"'] \arrow[dr, "n\in\sigma"] & \\ y_1 \arrow[dr, dotted, "p"'] & \sim & y_2 \arrow[dl, dotted, "q"] \\ & z & \end{tikzcd} \quad \text{implies} \quad \begin{tikzcd}[column sep=small] & x \arrow[dl, "\sigma\ni m"'] \arrow[dr, "n\in\sigma"] & \\ y_1 \arrow[dr, "\sigma\ni p"'] & \sim & y_2 \arrow[dl, "q\in\sigma"] \\ & z & \end{tikzcd}$$

where the dotted edges indicate edges in G.

2. backward compatibility preservation: dually, every asynchronous tile of the shape (12) in the 1-Player game G belongs to the subgraph G_σ of the strategy σ when its two cofinal transitions $p : y_1 \longrightarrow z$ and $q : y_2 \longrightarrow z$ are transitions in the subgraph G_σ.

These two properties ensure that the asynchronous graph G_σ is contractible and satisfies the cube property. Contractibility means that every two cofinal plays $s, t : * \twoheadrightarrow x$ of the strategy σ are equivalent modulo homotopy *inside* the asynchronous graph G_σ – that is, every intermediate play in the homotopy relation is an element of σ. Moreover, there is a simple reformulation as a set of plays of a positional strategy satisfying the two preservation properties: it is (essentially) a set of plays satisfying (1) a suitable cube property and (2) that $s \cdot m \in \sigma$ and $s \cdot n \in \sigma$ implies $s \cdot m \cdot p \in \sigma$ and $s \cdot n \cdot q \in \sigma$ when m and n are the coinitial moves of a tile (12). This characterization enables us to regard a positional strategy either as a set of plays, or as an asynchronous subgraph of the game.

4 Ingenuous Strategies in Asynchronous Games

A 2-Player game $(G, *, \lambda)$ is defined as an asynchronous graph $G = (V, E, \diamond)$ together with a distinguished initial position $*$, and a function $\lambda : E \to \{-1, +1\}$ which associates a *polarity* to every transition (or move) of the graph. Moreover, the equalities $\lambda(m) = \lambda(q)$ and $\lambda(n) = \lambda(p)$ are required to hold in every asynchronous tile (12) of the asynchronous graph G. The convention is that a move m is played by *Proponent* when $\lambda(m) = +1$ and by *Opponent* when $\lambda(m) = -1$.

A strategy σ is called *ingenuous* when it satisfies the following properties:

1. it is *positional*, and satisfies the backward and forward compatibility preservation properties of Section 3,
2. it is *deterministic*, in the following concurrent sense: every pair of coinitial moves $m : x \longrightarrow y_1$ and $n : x \longrightarrow y_2$ in the strategy σ where the move m is played by Proponent, induces an asynchronous tile (12) in the strategy σ. Diagrammatically,

$$\begin{tikzcd}[column sep=small] & x \arrow[dl, "\sigma\ni m"'] \arrow[dr, "n\in\sigma"] & \\ y_1 & & y_2 \end{tikzcd} \quad \text{implies} \quad \begin{tikzcd}[column sep=small] & x \arrow[dl, "\sigma\ni m"'] \arrow[dr, "n\in\sigma"] & \\ y_1 \arrow[dr, "\sigma\ni p"'] & \sim & y_2 \arrow[dl, "q\in\sigma"] \\ & z & \end{tikzcd}$$

3. it is *courteous*, in the following sense: every asynchronous tile (12) where the two moves $m : x \longrightarrow y_1$ and $p : y_1 \longrightarrow z$ are in the strategy σ, and where m is a Proponent move, is an asynchronous tile in the strategy σ. Diagrammatically,

$$\begin{tikzcd}
& x \arrow[dl, "\sigma\ni m"'] \arrow[dr, dotted, "n"] & & & & x \arrow[dl, "\sigma\ni m"'] \arrow[dr, "n\in\sigma"] & \\
y_1 \arrow[dr, "\sigma\ni p"'] & \sim & y_2 \arrow[dl, dotted, "q"] & \text{implies} & y_1 \arrow[dr, "\sigma\ni p"'] & \sim & y_2 \arrow[dl, "q\in\sigma"] \\
& z & & & & z &
\end{tikzcd}$$

when $\lambda(m) = +1$.

Note that, for simplicity, we express this series of conditions on strategies seen as asynchronous subgraphs. However, the conditions may be reformulated in a straightforward fashion on strategies defined as sets of plays. The forward and backward compatibility preservation properties of Section 3 ensure that the set of plays of the strategy σ reaching the same position x is regulated by a "causality order" on the moves occurring in these plays – which refines the "justification order" on moves (in the sense of pointer games) provided by the asynchronous game.

Our concurrent notion of determinism is not exactly the same as the usual notion of determinism in sequential games: in particular, a strategy may play several Proponent moves from a given position, as long as it converges later. Courtesy ensures that a strategy σ which accepts an Opponent move n *after* playing an independent Proponent move m, is ready to delay its own action, and to accept the move n *before* playing the move m. Together with the receptivity property introduced in Section 6, this ensures that the "causality order" on moves induced by such a strategy refines the underlying "justification order" of the game, by adding only order dependencies $m \preceq n$ where m is an Opponent move and n is a Proponent move. This adapts to the non-alternating setting the fact that, in alternating games, the causality order $p \preceq q$ provided by the view of an innocent strategy coincides with the justification order when p is Proponent and q is Opponent.

The experienced reader will notice that an ingenuous (and not necessarily receptive) strategy may add order dependencies $m \preceq n$ between two Opponent moves m and n. In the next Section, we will see that this reflects an unexpected property of concurrent strategies expressed as closure operators.

5 Ingenuous Strategies in Concurrent Games

In this section, we reformulate ingenuous strategies in asynchronous games as strategies in concurrent games. The assumption that our 2-Player games are played on contractible asynchronous graphs induces a partial order on the set of positions, defined by reachability: $x \leq y$ when $x \twoheadrightarrow y$. The concurrent game associated to an asynchronous game G is defined as the ideal completion of the partial order of positions reachable (in a finite number of steps) from the root $*$. So, the positions in the concurrent game are either *finite* when they are reachable from the root, or *infinite* when they are defined as an infinite directed subset of finite positions. The complete lattice D is then obtained by adding a top element $\top$ to the ideal completion. Considering infinite as well as finite positions introduces technicalities that we must confront in order to cope with infinite interactions, and to establish the functoriality property at the end of Section 6.

Now, we will reformulate ingenuous strategies in the asynchronous game G as *continuous* closure operators on the complete lattice D. By continuous, we mean that the closure operator preserves joins of directed subsets. Every closure operator σ on a complete lattice D induces a set of fixpoints:

$$\mathrm{fix}(\sigma) \quad = \quad \{\ x \in D \quad | \quad \sigma(x) = x\ \} \tag{14}$$

closed under arbitrary meets. Moreover, when the closure operator σ is continuous, the set $\mathrm{fix}(\sigma)$ is closed under joins of directed subsets. Conversely, every subset X of the complete lattice D closed under arbitrary meets defines a closure operator

$$\sigma \quad : \quad x \ \mapsto \ \bigwedge \{\ y \in X \quad | \quad x \leq y\ \} \tag{15}$$

which is continuous when the subset X is closed under joins of directed subsets. Moreover, the two translations (14) and (15) are inverse operations.

Now, every ingenuous strategy σ defines a set $\mathrm{halting}(\sigma)$ of *halting positions*. We say that a *finite* position x is *halting* when (1) the position is reached by the strategy σ and (2) there is no Proponent move $m : x \longrightarrow y$ in the strategy σ. The definition of a halting position can be extended to infinite positions by ideal completion: infinite positions are thus defined as downward-closed directed subsets $\hat{x}$ of finite positions. We do not provide the details here for lack of space. It can be shown that the set of halting positions of an ingenuous strategy σ is closed under arbitrary meets, and under joins of directed subsets. It thus defines a continuous closure operator, noted σ°, defined by (15). The closure operator σ° satisfies a series of additional properties:

1. The domain $\mathrm{dom}(\sigma^\circ)$ is closed under (arbitrary) compatible joins,
2. For every pair of positions $x, y \in \mathrm{dom}(\sigma^\circ)$ such that $x \leq y$, either $\sigma^\circ(x) = \sigma^\circ(y)$ or there exists an Opponent move $m : \sigma^\circ(x) \longrightarrow z$ such that $z \leq_P \sigma^\circ(z)$ and $\sigma^\circ(z) \leq \sigma^\circ(y)$.

Here, the *domain* $\mathrm{dom}(\sigma^\circ)$ of the closure operator σ° is defined as the set of positions $x \in D$ such that $\sigma^\circ(x) \neq \top$; and the Proponent reachability order $\leq_P$ refines the reachability order $\leq$ by declaring that $x \leq_P y$ means, for two finite positions x and y, that there exists a path $x \twoheadrightarrow y$ containing only Proponent moves; and then, by extending the definition of $\leq_P$ to all positions (either finite or infinite) in D by ideal completion.

Conversely, every continuous closure operator τ which satisfies the two additional properties mentioned above induces an ingenuous strategy σ in the following way. The *dynamic domain* of the closure operator τ is defined as the set of positions $x \in \mathrm{dom}(\tau)$ such that $x \leq_P \tau(x)$. So, a position x is in the dynamic domain of τ when the closure operator increases it in the proper way, that is, without using any Opponent move. The ingenuous strategy σ induced by the closure operator τ is then defined as the set of plays whose intermediate positions are all in the dynamic domain of τ. This defines an inverse to the operation $\sigma \mapsto \sigma^\circ$ from ingenuous strategies to continuous closure operators satisfying the additional properties 1 and 2. This pair of constructions thus provides a one-to-one correspondence between the ingenuous strategies and continuous closure operators satisfying the additional properties 1 and 2.

6 Innocent Strategies

Despite the one-to-one correspondence between ingenuous strategies and concurrent strategies described in Section 5, there is a subtle mismatch between the two notions – which fortunately disappears when ingenuity is refined into innocence. On the one hand, it is possible to construct a category $\mathcal{G}$ of asynchronous games and ingenuous strategies, where composition is defined by "parallel composition+hiding" on strategies seen as sets of plays. On the other hand, it is possible to construct a category $\mathcal{C}$ of concurrent games and concurrent strategies, defined in [3], where composition coincides with relational composition on the sets of fixpoints of closure operators. Unfortunately – and here comes the mismatch – the translation $\mathcal{G} \to \mathcal{C}$ described in Section 5 is not functorial, in the sense that it does not preserve composition of strategies. This is nicely illustrated by the example of the ingenuous strategy (11) whose composition with the right implementation of the strict conjunction (9) induces a deadlock. More conceptually, this phenomenon comes from the fact that the category $\mathcal{G}$ is compact closed: the tensor product $\otimes$ is identified with its dual Γ.

This motivates a strengthening of ingenuous strategies by a *scheduling criterion* which distinguishes the tensor product from its dual, and plays the role, in the non-alternating setting, of the *switching conditions* introduced by Abramsky and Jagadeesan for alternating games [2]. The criterion is sufficient to ensure that strategies do not deadlock during composition. In order to explain it here, we limit ourselves, for simplicity, to formulas of multiplicative linear logic (thus constructed using $\otimes$ and Γ and their units 1 and $\bot$) extended with the two lifting modalities $\uparrow$ and $\downarrow$. The tensor product, as well as its dual, are interpreted by the expected "asynchronous product" of asynchronous graphs: in particular, every play s of $A \otimes B$ may be seen as a pair of plays (s_A, s_B) of A and B modulo homotopy. The two connectives $\otimes$ and Γ are then distinguished by attaching a label $\otimes$ or Γ to every asynchronous tile (12) appearing in the game. Typically, an asynchronous tile (12) between a move m in A and a move n in B is labelled by $\otimes$ in the asynchronous game $A \otimes B$ and labelled by Γ in the asynchronous game $A \,\Gamma\, B$. The lifting modality $\uparrow$ (resp. $\downarrow$) is then interpreted as the operation of "lifting" a game with an initial Opponent (resp. Proponent) move. Note that there is a one-to-one relationship between the lifting modalities $\uparrow$ and $\downarrow$ appearing in the formula, and the moves of the asynchronous game G which denotes the formula. A nice aspect of our asynchronous approach is that we are able to formulate our scheduling criterion in two alternative but equivalent ways, each of them capturing a particular point of view on the correctness of proofs and strategies:

1. **a scheduling criterion** based on a switching as "before" $\olessthan$ or "after" $\ogreaterthan$ of every tensor product $\otimes$ in the underlying formula of linear logic. As explained in the introduction, the scheduling criterion requires that every path s in the strategy σ is equivalent modulo homotopy to a path t in the strategy σ which respects the scheduling indicated by the switching. This is captured diagrammatically by orienting every $\otimes$-tile as $\olessthan$ or $\ogreaterthan$ according to the switching, and by requiring that every play $s \in \sigma$ *normalizes* to a 2-dimensional normal form $t \in \sigma$ w.r.t. these semi-commutations [8] or standardization tiles [18].
2. **a directed acyclicity criterion** which reformulates the previous scheduling criterion along the lines of Girard's long trip criterion [13] and Danos-Regnier's

acyclicity criterion [10]. Every position x reached by an ingenuous strategy σ induces a partial order $\preceq$ on the moves appearing in the position x. Every relation $m \preceq n$ of the partial order induces a *jump* between the lifting modalities associated to the moves m and n in the formula. The acyclicity criterion then requires that every switching of the Γ connectives as "Left" or "Right" (that is, in the sense of Girard) induces a graph with no *directed* cycles.

The scheduling criterion ensures that the operation $\sigma \mapsto \sigma^\circ$ defines a *lax* functor, in the sense that every fixpoint of $\sigma^\circ; \tau^\circ$ is also a fixpoint of $(\sigma; \tau)^\circ$. Now, an ingenuous strategy σ is called *asynchronous* when it additionally satisfies the following *receptivity* property: for every play $s : * \twoheadrightarrow x$ and for every move $m : x \longrightarrow y$,

$$s \in \sigma \quad \text{and} \quad \lambda(m) = -1 \quad \text{implies} \quad s \cdot m \in \sigma.$$

The category $\mathcal{A}$ is then defined as follows: its objects are the asynchronous games equipped with $\otimes$-tiles and Γ-tiles, and its morphisms $A \to B$ are the asynchronous strategies of $A \multimap B$, defined as $A^* \,\Gamma\, B$, satisfying the scheduling criterion – where A^* is the asynchronous game A with Opponent and Proponent interchanged. The scheduling criterion ensures that the operation $\sigma \mapsto \sigma^\circ$ defines a strong monoidal functor $\mathcal{A} \to \mathcal{C}$ from the category $\mathcal{A}$ to the category $\mathcal{C}$ of concurrent games and concurrent strategies – thus extending the programme of [4,20] in the non-alternating setting.

The notion of asynchronous strategy is too liberal to capture the notion of innocent strategy, at least because there exist asynchronous strategies which are not definable as the interpretation of a proof of MLL extended with lifting modalities $\uparrow$ and $\downarrow$. The reason is that the scheduling criterion tests only for *directed* cycles, instead of the usual non-directed cycles. On the other hand, it should be noted that the directed acyclicity criterion coincides with the usual non-directed acyclicity criterion in the situation treated in [2] – that is, when the formula is purely multiplicative (i.e. contains no lifting modality), every variable X and $X^\perp$ is interpreted as a game with a Proponent and an Opponent move, and every axiom link is interpreted as a "bidirectional" copycat strategy. The full completeness result in [3] uses a similar directed acyclicity criterion for MALL. Hence, directed acyclicity is a fundamental, but somewhat hidden, concept of game semantics.

On the other hand, we would like to mirror the usual non-directed acyclicity criterion in our asynchronous and interactive framework. This leads us to another stronger scheduling criterion based on the idea that in every play s played by an asynchronous strategy σ, an Opponent move m and a Proponent move n appearing in the play, and directly related by the causality order $m \preceq n$ induced by σ can be played synchronously. We write $m \leftrightarrows n$ in that case, and say that two moves are synchronized in s when they lie in the same equivalence class generated by $\leftrightarrows$. A cluster of moves t in the play $s = m_1 \cdots m_k$ is then defined as a path $t = m_i \cdots m_{i+j}$ such that all the moves appearing in t are synchronized. Every play s in the strategy σ can be reorganized as a sequence of maximal clusters, using a standardization mechanism [18,19]. The resulting clustered play is unique, modulo permutation of clusters, noted $\sim_{OP}$. This relation generalizes to the non-alternating case the relation $\sim_{OP}$ introduced in [20]. This leads to

3. **a clustered scheduling criterion** based, just as previously, on a switching as "before" $\olessthan$ or "after" $\ogreaterthan$ of every tensor product $\otimes$ in the underlying formula of linear logic. The difference is that we ask that every clustered play s in the strategy σ may be reorganized modulo $\sim_{OP}$ as a clustered play which respects the scheduling indicated by the switching.

An asynchronous strategy is called innocent when it satisfies this stricter scheduling criterion. Although this tentative definition of innocence is fine conceptually, we believe that it has to be supported by further proof-theoretic investigations. An interesting aspect of our scheduling criteria is that they may be formulated in a purely diagrammatic and 2-dimensional way: in particular, the switching conditions are expressed here using the underlying logic MLL with lifting modalities $\uparrow$ and $\downarrow$ for clarity only, and may be easily reformulated diagrammatically.

Acknowledgments. We would like to thank Martin Hyland together with Pierre-Louis Curien, Claudia Faggian, Russ Harmer, Daniele Varacca, and Nobuko Yoshida for spontaneous and lively blackboard discussions.

References

1. Abramsky, S.: Sequentiality vs. concurrency in games and logic. In: MSCS (2003)
2. Abramsky, S., Jagadeesan, R.: Games and Full Completeness for Multiplicative Linear Logic. The Journal of Symbolic Logic 59(2), 543–574 (1994)
3. Abramsky, S., Melliès, P.-A.: Concurrent games and full completeness. In: LICS (1999)
4. Baillot, P., Danos, V., Ehrhard, T., Regnier, L.: Timeless Games. In: CSL (1997)
5. Bednarczyk, M.A.: Categories of asynchronous systems. PhD thesis (1988)
6. Berry, G.: Modèles complètement adéquats et stables des lambda-calculs typés. Thèse de Doctorat d'État, Université Paris VII (1979)
7. Bracho, F., Droste, M., Kuske, D.: Representation of computations in concurrent automata by dependence orders. Theoretical Computer Science 174(1-2), 67–96 (1997)
8. Clerbout, M., Latteux, M., Roos, Y.: The book of traces, ch. 12, pp. 487–552 (1995)
9. Curien, P.L., Faggian, C.: L-Nets, Strategies and Proof-Nets. In: CSL (2005)
10. Danos, V., Regnier, L.: The structure of multiplicatives. Archive for Mathematical Logic 28(3), 181–203 (1989)
11. Droste, M., Kuske, D.: Automata with concurrency relations – a survey. Advances in Logic, Artificial Intelligence and Robotics, pp. 152–172 (2002)
12. Ghica, D.R., Murawski, A.S.: Angelic Semantics of Fine-Grained Concurrency. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 211–225. Springer, Heidelberg (2004)
13. Girard, J.-Y.: Linear logic. TCS 50, 1–102 (1987)
14. Gonthier, G., Lévy, J.-J., Melliès, P.-A.: An abstract standardisation theorem. In: LICS (1992)
15. Goubault, É.: Geometry and Concurrency: A User's Guide. MSCS 10(4), 411–425 (2000)
16. Joyal, A.: Remarques sur la theorie des jeux à deux personnes. Gazette des Sciences Mathématiques du Quebec 1(4), 46–52 (1977)
17. Laird, J.: A game semantics of the asynchronous π-calculus. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, Springer, Heidelberg (2005)
18. Melliès, P.-A.: Axiomatic Rewriting 1: A diagrammatic standardization theorem. In: LNCS, pp. 554–638. Springer, Heidelberg (1992)

19. Melliès, P.-A.: Axiomatic rewriting 4: A stability theorem in rewriting theory. In: LICS (1998)
20. Melliès, P.-A.: Asynchronous games 2: the true concurrency of innocence. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, Springer, Heidelberg (2004)
21. Nielsen, M., Plotkin, G., Winskel, G.: Petri Nets, Event Structures and Domains, Part I. Theoretical Computer Science 13, 85–108 (1981)
22. Panangaden, P., Shanbhogue, V., Stark, E.W.: Stability and sequentiality in data flow networks. In: Paterson, M.S. (ed.) Automata, Languages and Programming. LNCS, vol. 443, pp. 253–264. Springer, Heidelberg (1990)
23. Sassone, V., Nielsen, M., Winskel, G.: Models for concurrency: Towards a classification. Theoretical Computer Science 170(1), 297–348 (1996)
24. Shields, M.W.: Concurrent Machines. The Computer Journal 28(5), 449–465 (1985)
25. Varacca, D., Yoshida, N.: Typed Event Structures and the π-calculus. In: MFPS (2006)
26. Winskel, G., Nielsen, M.: Models for concurrency. In: Handbook of Logic in Computer Science, vol. 3, pp. 1–148. Oxford University Press, Oxford (1995)

Bisimulation and Logical Preservation for Continuous-Time Markov Decision Processes

Martin R. Neuhäußer[1,2] and Joost-Pieter Katoen[1,2]

[1] Software Modeling and Verification Group
RWTH Aachen University, Germany
[2] Formal Methods and Tools Group
University of Twente, The Netherlands
{neuhaeusser,katoen}@cs.rwth-aachen.de

Abstract. This paper introduces strong bisimulation for continuous-time Markov decision processes (CTMDPs), a stochastic model which allows for a nondeterministic choice between exponential distributions, and shows that bisimulation preserves the validity of CSL. To that end, we interpret the semantics of CSL—a stochastic variant of CTL for continuous-time Markov chains—on CTMDPs and show its measure-theoretic soundness. The main challenge faced in this paper is the proof of logical preservation that is substantially based on measure theory.

1 Introduction

Discrete–time probabilistic models, in particular Markov decision processes (MDP) [20], are used in various application areas such as randomized distributed algorithms and security protocols. A plethora of results in the field of concurrency theory and verification are known for MDPs. Efficient model–checking algorithms exist for probabilistic variants of CTL [9,11], linear–time [30] and long–run properties [15], process algebraic formalisms for MDPs have been developed and bisimulation is used to minimize MDPs prior to analysis [18].

In contrast, CTMDPs [26], a continuous–time variant of MDPs, where state residence times are exponentially distributed, have received scant attention. Whereas in MDPs nondeterminism occurs between discrete probability distributions, in CTMDPs the choice between various exponential distributions is nondeterministic. In case all exponential delays are uniquely determined, a continuous–time Markov chain (CTMC) results, a widely studied model in performance and dependability analysis.

This paper proposes strong bisimulation on CTMDPs—this notion is a conservative extension of bisimulation on CTMCs [13]—and investigates which kind of logical properties this preserves. In particular, we show that bisimulation preserves the validity of CSL [3,5], a well–known logic for CTMCs. To that end, we provide a semantics of CSL on CTMDPs which is in fact obtained in a similar way as the semantics of PCTL on MDPs [9,11]. We show the semantic soundness of the logic using measure–theoretic arguments, and prove that bisimilar states preserve full CSL. Although this result is perhaps not surprising, its proof is non–trivial and strongly relies on measure–theoretic aspects. It shows that reasoning about CTMDPs, as witnessed also by [31,7,10] is not straightforward. As for MDPs, CSL equivalence does not coincide with bisimulation as only maximal and minimal probabilities can be logically expressed.

Apart from the theoretical contribution, we believe that the results of this paper have wider applicability. CTMDPs are the semantic model of stochastic Petri nets [14] that exhibit confusion, stochastic activity networks [28] (where absence of nondeterminism is validated by a "well–specified" check), and is strongly related to interactive Markov chains which are used to provide compositional semantics to process algebras [19] and dynamic fault trees [12]. Besides, CTMDPs have practical applicability in areas such as stochastic scheduling [17,1] and dynamic power management [27]. Our interest in CTMDPs is furthermore stimulated by recent results on abstraction—where the introduction of nondeterminism is the key principle—of CTMCs [21] in the context of probabilistic model checking.

In our view, it is a challenge to study this continuous–time stochastic model in greater depth. This paper is a small, though important, step towards a better understanding of CTMDPs. More details and all proofs can be found in [25].

2 Continuous-Time Markov Decision Processes

Continuous-time Markov decision processes extend continuous-time Markov chains by nondeterministic choices. Therefore each transition is labelled with an action referring to the nondeterministic choice and the rate of a negative exponential distribution which determines the transition's delay:

Definition 1 (Continuous-time Markov decision process). *A tuple* $\mathcal{C} = (\mathcal{S}, Act, \mathbf{R}, AP, L)$ *is a* labelled continuous-time Markov decision process *if* $\mathcal{S}$ *is a finite, nonempty set of* states*,* Act *a finite, nonempty set of* actions *and* $\mathbf{R} : \mathcal{S} \times Act \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ *a three-dimensional* rate matrix*. Further,* AP *is a finite set of* atomic propositions *and* $L : \mathcal{S} \to 2^{AP}$ *is a* state labelling function.

The set of actions that are enabled in a state $s \in \mathcal{S}$ is denoted $Act(s) := \{\alpha \in Act \mid \exists s' \in \mathcal{S}.\ \mathbf{R}(s, \alpha, s') > 0\}$. A CTMDP is *well-formed* if $Act(s) \neq \emptyset$ for all $s \in \mathcal{S}$, that is, if every state has at least one outgoing transition. Note that this can easily be established for any CTMDP by adding self-loops.

Example 1. When entering state s_1 of the CTMDP in Fig. 1 (without state labels) one action from the set of enabled actions $Act(s_1) = \{\alpha, \beta\}$ is chosen nondeterministically, say α. Next, the rate of the α-transition determines its exponentially distributed delay. Hence for a single transition, the probability to go from s_1 to s_3 within time t is $1 - e^{-\mathbf{R}(s_1,\alpha,s_3)t} = 1 - e^{-0.1t}$.

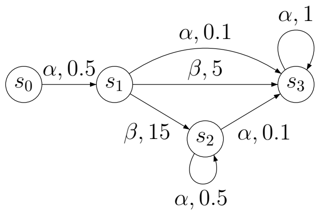

$$\begin{tikzcd}
s_0 \arrow[r, "{\alpha, 0.5}"] & s_1 \arrow[rr, bend left, "{\alpha, 0.1}"] \arrow[rr, "{\beta, 5}"] \arrow[dr, "{\beta, 15}"'] & & s_3 \arrow[loop above, "{\alpha, 1}"] \\
& & s_2 \arrow[ur, "{\alpha, 0.1}"'] \arrow[loop below, "{\alpha, 0.5}"] &
\end{tikzcd}$$

Fig. 1. Example of a CTMDP

If multiple outgoing transitions exist for the chosen action, they compete according to their exponentially distributed delays: In Fig. 1 such a *race condition* occurs if action β is chosen in state s_1. In this situation, two β-transitions (to s_2 and s_3) with rates $\mathbf{R}(s_1,\beta,s_2) = 15$ and $\mathbf{R}(s_1,\beta,s_3) = 5$ become available and state s_1 is left as soon as the first transition's delay expires. Hence the sojourn time in state s_1 is distributed according to the minimum of both exponential distributions, i.e. with rate $\mathbf{R}(s_1,\beta,s_2) + \mathbf{R}(s_1,\beta,s_3) = 20$. In general, $E(s,\alpha) := \sum_{s'\in\mathcal{S}} \mathbf{R}(s,\alpha,s')$ is the *exit rate* of state s under action α. Then $\mathbf{R}(s_1,\beta,s_2)/E(s_1,\beta) = 0.75$ is the probability to move with β from s_1 to s_2, i.e. the probability that the delay of the β-transition to s_2 expires first. Formally, the *discrete branching probability* is $\mathbf{P}(s,\alpha,s') := \frac{\mathbf{R}(s,\alpha,s')}{E(s,\alpha)}$ if $E(s,\alpha) > 0$ and 0 otherwise. By $\mathbf{R}(s,\alpha,Q) := \sum_{s'\in Q} \mathbf{R}(s,\alpha,s')$ we denote the total rate to states in $Q \subseteq \mathcal{S}$.

Definition 2 (Path). *Let $\mathcal{C} = (\mathcal{S}, Act, \mathbf{R}, AP, L)$ be a CTMDP. $Paths^n(\mathcal{C}) := \mathcal{S} \times (Act \times \mathbb{R}_{\geq 0} \times \mathcal{S})^n$ is the set of paths of length n in $\mathcal{C}$; the set of finite paths in $\mathcal{C}$ is defined by $Paths^\star(\mathcal{C}) = \bigcup_{n\in\mathbb{N}} Paths^n$ and $Paths^\omega(\mathcal{C}) := (\mathcal{S} \times Act \times \mathbb{R}_{\geq 0})^\omega$ is the set of infinite paths in $\mathcal{C}$. $Paths(\mathcal{C}) := Paths^\star(\mathcal{C}) \cup Paths^\omega(\mathcal{C})$ denotes the set of all paths in $\mathcal{C}$.*

We write *Paths* instead of $Paths(\mathcal{C})$ whenever $\mathcal{C}$ is clear from the context. Paths are denoted $\pi = s_0 \xrightarrow{\alpha_0,t_0} s_1 \xrightarrow{\alpha_1,t_1} \cdots \xrightarrow{\alpha_{n-1},t_{n-1}} s_n$ where $|\pi|$ is the length of π. Given a finite path $\pi \in Paths^n$, $\pi\downarrow$ is the last state of π. For $n < |\pi|$, $\pi[n] := s_n$ is the n-th state of π and $\delta(\pi,n) := t_n$ is the time spent in state s_n. Further, $\pi[i..j]$ is the path-infix $s_i \xrightarrow{\alpha_i,t_i} s_{i+1} \xrightarrow{\alpha_{i+1},t_{i+1}} \cdots \xrightarrow{\alpha_{j-1},t_{j-1}} s_j$ of π for $i < j \leq |\pi|$. We write $\xrightarrow{\alpha,t} s'$ for a transition with action α at time point t to a successor state s'. The extension of a path π by a transition m is denoted $\pi \circ m$. Finally, $\pi@t$ is the state occupied in π at time point $t \in \mathbb{R}_{\geq 0}$, i.e. $\pi@t := \pi[n]$ where n is the smallest index such that $\sum_{i=0}^n t_i > t$.

Note that Def. 2 does not impose any semantic restrictions on paths, i.e. the set *Paths* usually contains paths which do not exist in the underlying CTMDP. However, the following definition of the probability measure (Def. 4) justifies this as it assigns probability zero to those sets of paths.

2.1 The Probability Space

In probability theory (see [2]), a *field* of sets $\mathfrak{F} \subseteq 2^\Omega$ is a family of subsets of a set Ω which contains the empty set and is closed under complement and finite union. A field $\mathfrak{F}$ is a *σ-field*[1] if it is also closed under countable union, i.e. if for all countable families $\{A_i\}_{i\in I}$ of sets $A_i \in \mathfrak{F}$ it holds $\bigcup_{i\in I} A_i \in \mathfrak{F}$. Any subset A of Ω which is in $\mathfrak{F}$ is called *measurable*. To measure the probability of sets of paths, we define a σ-field of sets of *combined transitions* which we later use to define σ-fields of sets of finite and infinite paths: For CTMDP $\mathcal{C} = (\mathcal{S}, Act, \mathbf{R}, AP, L)$, the set of combined transitions is $\Omega = Act \times \mathbb{R}_{\geq 0} \times \mathcal{S}$. As $\mathcal{S}$ and Act are finite, the corresponding σ-fields are $\mathfrak{F}_{Act} := 2^{Act}$ and $\mathfrak{F}_{\mathcal{S}} := 2^{\mathcal{S}}$; further, $Distr(Act)$ and $Distr(\mathcal{S})$ denote the sets of probability distributions on $\mathfrak{F}_{Act}$ and $\mathfrak{F}_{\mathcal{S}}$. Any

[1] In the literature [22], σ-fields are also called σ-algebras.

combined transition occurs at some time point $t \in \mathbb{R}_{\geq 0}$ so that we can use the Borel σ-field $\mathfrak{B}(\mathbb{R}_{\geq 0})$ to measure the corresponding subsets of $\mathbb{R}_{\geq 0}$.

A Cartesian product is a *measurable rectangle* if its constituent sets are elements of their respective σ-fields, i.e. the set $A \times T \times S$ is a measurable rectangle if $A \in \mathfrak{F}_{Act}$, $T \in \mathfrak{B}(\mathbb{R}_{\geq 0})$ and $S \in \mathfrak{F}_{\mathcal{S}}$. We use $\mathfrak{F}_{Act} \times \mathfrak{B}(\mathbb{R}_{\geq 0}) \times \mathfrak{F}_{\mathcal{S}}$ to denote the set of all measurable rectangles[2]. It generates the desired σ-field $\mathfrak{F}$ of sets of combined transitions, i.e. $\mathfrak{F} := \sigma\big(\mathfrak{F}_{Act} \times \mathfrak{B}(\mathbb{R}_{\geq 0}) \times \mathfrak{F}_{\mathcal{S}}\big)$.

Now $\mathfrak{F}$ may be used to infer the σ-fields $\mathfrak{F}_{Paths^n}$ of sets of paths of length n: $\mathfrak{F}_{Paths^n}$ is generated by the set of measurable (path) rectangles, i.e. $\mathfrak{F}_{Paths^n} := \sigma\big(\{S_0 \times M_0 \times \cdots \times M_n \mid S_0 \in \mathfrak{F}_{\mathcal{S}}, M_i \in \mathfrak{F}, 0 \leq i \leq n\}\big)$. The σ-field of sets of infinite paths is obtained using the cylinder-set construction [2]: A set C^n of paths of length n is called a *cylinder base*; it induces the infinite *cylinder* $C_n = \{\pi \in Paths^\omega \mid \pi[0..n] \in C^n\}$. A cylinder C_n is *measurable* if $C^n \in \mathfrak{F}_{Paths^n}$; C_n is a rectangle if $C^n = S_0 \times A_0 \times T_0 \times \cdots \times A_{n-1} \times T_{n-1} \times S_n$ and $S_i \subseteq \mathcal{S}$, $A_i \subseteq Act$ and $T_i \subseteq \mathbb{R}_{\geq 0}$. It is a *measurable rectangle*, if $S_i \in \mathfrak{F}_{\mathcal{S}}$, $A_i \in \mathfrak{F}_{Act}$ and $T_i \in \mathfrak{B}(\mathbb{R}_{\geq 0})$. Finally, the σ-field of sets of infinite paths is defined as $\mathfrak{F}_{Paths^\omega} := \sigma\big(\bigcup_{n=0}^{\infty} \{C_n \mid C^n \in \mathfrak{F}_{Paths^n}\}\big)$.

2.2 The Probability Measure

To define a semantics for CTMDP we use schedulers[3] to resolve the nondeterministic choices. Thereby we obtain probability measures on the probability spaces defined above. A scheduler quantifies the probability of the next action based on the history of the system: If state s is reached via finite path π, the scheduler yields a probability distribution over $Act(\pi{\downarrow})$. The type of schedulers we use is the class of measurable timed history-dependent randomized schedulers [31]:

Definition 3 (Measurable scheduler). *Let $\mathcal{C}$ be a CTMDP with action set Act. A mapping $\mathcal{D} : Paths^\star \times \mathfrak{F}_{Act} \to [0,1]$ is a* measurable scheduler *if $\mathcal{D}(\pi, \cdot) \in Distr(Act(\pi{\downarrow}))$ for all $\pi \in Paths^\star$ and the functions $\mathcal{D}(\cdot, A) : Paths^\star \to [0,1]$ are measurable for all $A \in \mathfrak{F}_{Act}$. THR denotes the set of measurable schedulers.*

In Def. 3, the measurability condition states that for any $B \in \mathfrak{B}([0,1])$ and $A \in \mathfrak{F}_{Act}$ the set $\{\pi \in Paths^\star \mid \mathcal{D}(\pi, A) \in B\} \in \mathfrak{F}_{Paths^\star}$, see [31]. In the following, note that $\mathcal{D}(\pi, \cdot)$ is a probability measure with support $\subseteq Act(\pi{\downarrow})$; further $\mathbf{P}(s, \alpha, \cdot) \in Distr(\mathcal{S})$ if $\alpha \in Act(s)$. Let $\eta_{E(\pi\downarrow,\alpha)}(t) := E(\pi{\downarrow}, \alpha) \cdot e^{-E(\pi\downarrow,\alpha)t}$ denote the probability density function of the negative exponential distribution with parameter $E(\pi{\downarrow}, \alpha)$. To derive a probability measure on $\mathfrak{F}_{Paths^\omega}$, we first define a probability measure on $(\Omega, \mathfrak{F})$: For history $\pi \in Paths^\star$, let $\mu_{\mathcal{D}}(\pi, \cdot) : \mathfrak{F} \to [0,1]$ such that

$$\mu_{\mathcal{D}}(\pi, M) := \int_{Act} \mathcal{D}(\pi, d\alpha) \int_{\mathbb{R}_{\geq 0}} \eta_{E(\pi\downarrow,\alpha)}(dt) \int_{\mathcal{S}} \mathbf{I}_M(\alpha, t, s)\ \mathbf{P}(\pi{\downarrow}, \alpha, ds).$$

Then $\mu_{\mathcal{D}}(\pi, \cdot)$ defines a probability measure on $\mathfrak{F}$ where the indicator function $\mathbf{I}_M(\alpha, t, s) := 1$ if the combined transition $(\alpha, t, s) \in M$ and 0 otherwise [31]. For a measurable rectangle $A \times T \times S' \in \mathfrak{F}$ we obtain

[2] Despite notation, $\mathfrak{F}_{Act} \times \mathfrak{B}(\mathbb{R}_{\geq 0}) \times \mathfrak{F}_{\mathcal{S}}$ is not a Cartesian product.

[3] Schedulers are also called policies or adversaries in the literature.

$$\mu_{\mathcal{D}}(\pi, A \times T \times S') = \sum_{\alpha \in A} \mathcal{D}(\pi, \{\alpha\}) \cdot \mathbf{P}(\pi{\downarrow}, \alpha, S') \cdot \int_T E(\pi{\downarrow}, \alpha) \cdot e^{-E(\pi{\downarrow},\alpha)t} dt. \qquad (1)$$

Intuitively, $\mu_{\mathcal{D}}(\pi, A \times T \times S')$ is the probability to leave $\pi{\downarrow}$ via some action in A within time interval T to a state in S'. To extend this to a probability measure on paths, we now assume an *initial distribution* $\nu \in \mathit{Distr}(\mathcal{S})$ for the probability to start in a certain state s; instead of $\nu(\{s\})$ we also write $\nu(s)$.

Definition 4 (Probability measure [31]). *For initial distribution $\nu \in \mathit{Distr}(\mathcal{S})$ the probability measure on $\mathfrak{F}_{\mathit{Paths}^n}$ is defined inductively:*

$$\begin{aligned} &Pr^0_{\nu,\mathcal{D}} : \mathfrak{F}_{\mathit{Paths}^0} \to [0,1] \ : \Pi \mapsto \sum_{s \in \Pi} \nu(s) \quad \textit{and for } n > 0 \\ &Pr^n_{\nu,\mathcal{D}} : \mathfrak{F}_{\mathit{Paths}^n} \to [0,1] \ : \Pi \mapsto \int_{\mathit{Paths}^{n-1}} Pr^{n-1}_{\nu,\mathcal{D}}(d\pi) \int_{\Omega} \mathbf{I}_{\Pi}(\pi \circ m)\, \mu_{\mathcal{D}}(\pi, dm). \end{aligned}$$

By Def. 4 we obtain measures on all σ-fields $\mathfrak{F}_{\mathit{Paths}^n}$. This extends to a measure on $(\mathit{Paths}^\omega, \mathfrak{F}_{\mathit{Paths}^\omega})$ as follows: First, note that any measurable cylinder can be represented by a base of finite length, i.e. $C_n = \{\pi \in \mathit{Paths}^\omega \mid \pi[0..n] \in C^n\}$. Now the measures $Pr^n_{\nu,\mathcal{D}}$ on $\mathfrak{F}_{\mathit{Paths}^n}$ extend to a unique probability measure $Pr^\omega_{\nu,\mathcal{D}}$ on $\mathfrak{F}_{\mathit{Paths}^\omega}$ by defining $Pr^\omega_{\nu,\mathcal{D}}(C_n) = Pr^n_{\nu,\mathcal{D}}(C^n)$. Although any measurable rectangle with base C^m can equally be represented by a higher-dimensional base (more precisely, if $m < n$ and $C^n = C^m \times \Omega^{n-m}$ then $C_n = C_m$), the Ionescu–Tulcea extension theorem [2] is applicable due to the inductive definition of the measures $Pr^n_{\nu,\mathcal{D}}$ and assures the extension to be well defined and unique.

Definition 4 inductively *appends* transition triples to the path prefixes of length n to obtain a measure on sets of paths of length $n+1$. In the proof of Theorem 3, we use an equivalent characterization that constructs paths reversely, i.e. paths of length $n+1$ are obtained from paths of length n by concatenating an *initial triple* from the set $\mathcal{S} \times \mathit{Act} \times \mathbb{R}_{\geq 0}$ to the suffix of length n:

Definition 5 (Initial triples). *Let $\mathcal{C} = (\mathcal{S}, \mathit{Act}, \mathbf{R}, \mathit{AP}, L)$ be a CTMDP, $\nu \in \mathit{Distr}(\mathcal{S})$ and $\mathcal{D}$ a scheduler. Then the measure $\mu_{\nu,\mathcal{D}} : \mathfrak{F}_{\mathcal{S}\times \mathit{Act} \times \mathbb{R}_{\geq 0}} \to [0,1]$ on sets I of* initial triples *(s, α, t) is defined as*

$$\mu_{\nu,\mathcal{D}}(I) = \int_{\mathcal{S}} \nu(ds) \int_{\mathit{Act}} \mathcal{D}(s, d\alpha) \int_{\mathbb{R}_{\geq 0}} \mathbf{I}_I(s, \alpha, t) \ \ \eta_{E(s,\alpha)}(dt).$$

This allows to decompose a path $\pi = s_0 \xrightarrow{\alpha_0, t_0} \cdots \xrightarrow{\alpha_{n-1}, t_{n-1}} s_n$ into an initial triple $i = (s_0, \alpha_0, t_0)$ and the path suffix $\pi[1..n]$. For this to be measure preserving, a new $\nu_i \in \mathit{Distr}(\mathcal{S})$ is defined based on the original initial distribution ν of $Pr^n_{\nu,\mathcal{D}}$ on $\mathfrak{F}_{\mathit{Paths}^n}$ which reflects the fact that state s_0 has already been left with action α_0 at time t_0. Hence ν_i is the initial distribution for the suffix-measure on $\mathfrak{F}_{\mathit{Paths}^{n-1}}$. Similarly, a scheduler $\mathcal{D}_i$ is defined which reproduces the decisions of the original scheduler $\mathcal{D}$ given that the first i-step is already taken. Hence $\mathrm{Pr}^{n-1}_{\nu_i,\mathcal{D}_i}$ is the adjusted probability measure on $\mathfrak{F}_{\mathit{Paths}^{n-1}}$ given ν_i and $\mathcal{D}_i$.

Lemma 1. *For $n \geq 1$ let $I \times \Pi \in \mathfrak{F}_{\mathit{Paths}^n}$ be a measurable rectangle, where $I \in \mathfrak{F}_{\mathcal{S}} \times \mathfrak{F}_{\mathit{Act}} \times \mathfrak{B}(\mathbb{R}_{\geq 0})$. For $i = (s, \alpha, t) \in I$, let $\nu_i := \mathbf{P}(s, \alpha, \cdot)$ and $\mathcal{D}_i(\pi) := \mathcal{D}(i \circ \pi)$. Then $Pr^n_{\nu,\mathcal{D}}(I \times \Pi) = \int_I Pr^{n-1}_{\nu_i,\mathcal{D}_i}(\Pi)\, \mu_{\nu,\mathcal{D}}(di)$.*

Proof. By induction on n:

- induction start ($n = 1$): Let $\Pi \in \mathfrak{F}_{Paths^0}$, i.e. $\Pi \subseteq \mathcal{S}$.

$$\begin{aligned}
Pr^1_{\nu,\mathcal{D}}(I \times \Pi) &= \int_{Paths^0} Pr^0_{\nu,\mathcal{D}}(d\pi) \int_{\Omega} \mathbf{I}_{I\times\Pi}(\pi \circ m) \;\; \mu_{\mathcal{D}}(\pi, dm) && \text{(* Definition 4 *)}\\
&= \int_{\mathcal{S}} \nu(ds_0) \int_{\Omega} \mathbf{I}_{I\times\Pi}(s_0 \circ m) \;\; \mu_{\mathcal{D}}(s_0, dm) && \text{(* } Paths^0 = \mathcal{S} \text{ *)}\\
&= \int_{\mathcal{S}} \nu(ds_0) \int_{Act} \mathcal{D}(s_0, d\alpha_0) \int_{\mathbb{R}_{\geq 0}} \eta_{E(s_0,\alpha_0)}(dt_0) \int_{\mathcal{S}} \mathbf{I}_{I\times\Pi}(s_0 \xrightarrow{\alpha_0,t_0} s_1) \;\; \mathbf{P}(s_0, \alpha_0, ds_1)\\
&= \int_{I} \mu_{\nu,\mathcal{D}}(ds_0, d\alpha_0, dt_0) \int_{\mathcal{S}} \mathbf{I}_{\Pi}(s_1) \;\; \mathbf{P}(s_0, \alpha_0, ds_1) && \text{(* definition of } \mu_{\nu,\mathcal{D}} \text{*)}\\
&= \int_{I} \mu_{\nu,\mathcal{D}}(di) \int_{\mathcal{S}} \mathbf{I}_{\Pi}(s_1) \;\; \nu_i(ds_1) && \text{(* } i = (s_0, \alpha_0, t_0) \text{ *)}\\
&= \int_{I} Pr^0_{\nu_i,\mathcal{D}_i}(\Pi) \;\; \mu_{\nu,\mathcal{D}}(di). && \text{(* Definition 4 *)}
\end{aligned}$$

- induction step ($n > 1$): Let $I \times \Pi \times M$ be a measurable rectangle in $\mathfrak{F}_{Paths^{n+1}}$ such that $I \in \mathfrak{F}_{\mathcal{S}} \times \mathfrak{F}_{Act} \times \mathfrak{B}(\mathbb{R}_{\geq 0})$ is a set of initial triples, $\Pi \in \mathfrak{F}_{Paths^{n-1}}$ and $M \in \mathfrak{F}$ is a set of combined transitions. Using the induction hypothesis $Pr^n_{\nu,\mathcal{D}}(I \times \Pi) = \int_I Pr^{n-1}_{\nu_i,\mathcal{D}_i}(\Pi) \;\; \mu_{\nu,\mathcal{D}}(di)$ we derive:

$$\begin{aligned}
Pr^{n+1}_{\nu,\mathcal{D}}(I \times \Pi \times M) &= \int_{I\times\Pi} \mu_{\mathcal{D}}(\pi, M) \;\; Pr^n_{\nu,\mathcal{D}}(d\pi) && \text{(* Definition 4 *)}\\
&= \int_{I\times\Pi} \mu_{\mathcal{D}}(i \circ \pi', M) \;\; Pr^n_{\nu,\mathcal{D}}(d(i \circ \pi')) && \text{(* } \pi \simeq i \circ \pi' \text{ *)}\\
&= \int_{I} \int_{\Pi} \mu_{\mathcal{D}}(i \circ \pi', M) \;\; Pr^{n-1}_{\nu_i,\mathcal{D}_i}(d\pi') \;\; \mu_{\nu,\mathcal{D}}(di) && \text{(* ind. hypothesis *)}\\
&= \int_{I} \int_{\Pi} \mu_{\mathcal{D}_i}(\pi', M) \;\; Pr^{n-1}_{\nu_i,\mathcal{D}_i}(d\pi') \;\; \mu_{\nu,\mathcal{D}}(di) && \text{(* definition of } \mathcal{D}_i \text{ *)}\\
&= \int_{I} Pr^n_{\nu_i,\mathcal{D}_i}(\Pi \times M) \;\; \mu_{\nu,\mathcal{D}}(di). && \text{(* Definition 4 *)}
\end{aligned}$$

□

A class of pathological paths that are not ruled out by Def. 2 are infinite paths whose duration converges to some real constant, i.e. paths that visit infinitely many states in a finite amount of time. For $n = 0, 1, 2, \ldots$, an increasing sequence $r_n \in \mathbb{R}_{\geq 0}$ is *Zeno* if it converges to a positive real number. For example, $r_n := \sum_{i=1}^{n} \frac{1}{2^n}$ converges to 1, hence is Zeno. The following theorem justifies to rule out such Zeno behaviour:

Theorem 1 (Converging paths theorem). *The probability measure of the set of converging paths is zero.*

Proof. Let $ConvPaths := \left\{ s_0 \xrightarrow{\alpha_0,t_0} s_1 \xrightarrow{\alpha_1,t_1} \cdots \mid \sum_{i=0}^{n} t_i \text{ converges} \right\}$. Then for $\pi \in ConvPaths$ the sequence t_i converges to 0. Thus there exists $k \in \mathbb{N}$ such that $t_i \leq 1$ for all $i \geq k$. Hence $ConvPaths \subseteq \bigcup_{k\in\mathbb{N}} \mathcal{S} \times \Omega^k \times (Act \times [0,1] \times \mathcal{S})^{\omega}$. Similar to [5, Prop. 1], it can be shown that $Pr^{\omega}_{\nu,\mathcal{D}}\left(\mathcal{S} \times \Omega^k \times (Act \times [0,1] \times \mathcal{S})^{\omega}\right) = 0$ for all $k \in \mathbb{N}$. Thus also $Pr^{\omega}_{\nu,\mathcal{D}}\left(\bigcup_{k\in\mathbb{N}} \mathcal{S} \times \Omega^k \times (Act \times [0,1] \times \mathcal{S})^{\omega}\right) = 0$. *ConvPaths* is a subset of a set of measure zero; hence, on $\mathfrak{F}_{Paths^\omega}$ completed[4] w.r.t. $Pr^{\omega}_{\nu,\mathcal{D}}$ we obtain $Pr^{\omega}_{\nu,\mathcal{D}}(ConvPaths) = 0$. □

3 Strong Bisimulation

Strong bisimulation [8,23] is an equivalence on the set of states of a CTMDP which relates two states if they are equally labelled and exhibit the same stepwise behaviour. As shown in Theorem 4, strong bisimilarity allows one to aggregate the state space while preserving transient and long run measures.

In the following we denote the equivalence class of s under equivalence $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ by $[s]_{\mathcal{R}} = \{s' \in \mathcal{S} \mid (s, s') \in \mathcal{R}\}$; if $\mathcal{R}$ is clear from the context we also write $[s]$. Further, $\mathcal{S}_{\mathcal{R}} := \{[s]_{\mathcal{R}} \mid s \in \mathcal{S}\}$ is the quotient space of $\mathcal{S}$ under $\mathcal{R}$.

Definition 6 (Strong bisimulation relation). *Let $\mathcal{C} = (\mathcal{S}, Act, \mathbf{R}, AP, L)$ be a CTMDP. An equivalence $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is a* strong bisimulation relation *if $L(u) = L(v)$ for all $(u, v) \in \mathcal{R}$ and $\mathbf{R}(u, \alpha, C) = \mathbf{R}(v, \alpha, C)$ for all $\alpha \in Act$ and all $C \in \mathcal{S}_{\mathcal{R}}$.*

Two states u and v are strongly bisimilar *($u \sim v$) if there exists a strong bisimulation relation $\mathcal{R}$ such that $(u, v) \in \mathcal{R}$.* Strong bisimilarity *is the union of all strong bisimulation relations.*

Formally, $\sim \,=\, \{(u, v) \in \mathcal{S} \times \mathcal{S} \mid \exists$ str. bisimulation rel. $\mathcal{R}$ with $(u, v) \in \mathcal{R}\}$ defines strong bisimilarity which itself is (the largest) strong bisimulation relation.

Definition 7 (Quotient). *Let $\mathcal{C} = (\mathcal{S}, Act, \mathbf{R}, AP, L)$ be a CTMDP. Then $\tilde{\mathcal{C}} := (\tilde{\mathcal{S}}, Act, \tilde{\mathbf{R}}, AP, \tilde{L})$ where $\tilde{\mathcal{S}} := \mathcal{S}_{\sim}$, $\tilde{\mathbf{R}}([s], \alpha, C) := \mathbf{R}(s, \alpha, C)$ and $\tilde{L}([s]) := L(s)$ for all $s \in \mathcal{S}$, $\alpha \in Act$ and $C \in \tilde{\mathcal{S}}$ is the* quotient *of $\mathcal{C}$ under strong bisimilarity.*

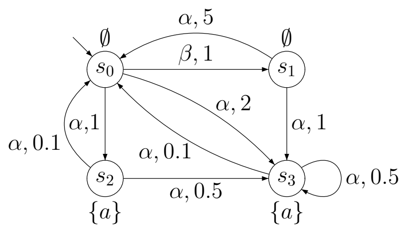
To distinguish between a CTMDP $\mathcal{C}$ and its quotient, let $\tilde{\mathbf{P}}$ denote the quotient's discrete branching probabilities and $\tilde{E}$ its exit rates. Note however, that exit rates and branching probabilities are preserved by strong bisimilarity, i.e. $E(s, \alpha) = \tilde{E}([s], \alpha)$ and $\tilde{\mathbf{P}}([s], \alpha, [t]) = \sum_{t' \in [t]} \mathbf{P}(s, \alpha, t')$ for $\alpha \in Act$ and $s, t \in \mathcal{S}$.

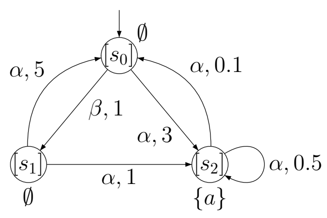
Example 2. Consider the CTMDP over the set $AP = \{a\}$ of atomic propositions in Fig. 2(a). Its quotient under strong bisimilarity is outlined in Fig. 2(b).

4 Continuous Stochastic Logic

Continuous stochastic logic [3,5] is a state-based logic to reason about continuous-time Markov chains. In this context, its formulas characterize strong bisimilarity [16] as defined in [5]; moreover, strongly bisimilar states satisfy the same CSL formulas [5]. In this paper, we extend CSL to CTMDPs along the lines of [6] and further introduce a long-run average operator [15]. Our semantics is based on ideas from [9,11] where variants of PCTL are extended to (discrete time) MDPs.

[4] We may assume $\mathfrak{F}_{Paths^\omega}$ to be complete, see [2, p. 18ff].





(a) CTMDP $\mathcal{C}$ (b) Quotient $\tilde{\mathcal{C}}$

Fig. 2. Quotient under strong bisimilarity

4.1 Syntax and Semantics

Definition 8 (CSL syntax). *For $a \in AP$, $p \in [0,1]$, $I \subseteq \mathbb{R}_{\geq 0}$ a nonempty interval and $\sqsubseteq \in \{<, \leq, \geq, >\}$,* CSL state *and* CSL path formulas *are defined by*

$$\Phi ::= \ a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \forall^{\sqsubseteq p}\varphi \mid \mathsf{L}^{\sqsubseteq p}\Phi \qquad \textit{and} \qquad \varphi ::= \ \mathsf{X}^I\Phi \mid \Phi\mathsf{U}^I\Phi.$$

The Boolean connectives $\vee$ and $\rightarrow$ are defined as usual; further we extend the syntax by deriving the timed modal operators "eventually" and "always" using the equalities $\Diamond^I\Phi \equiv \mathsf{tt}\mathsf{U}^I\Phi$ and $\Box^I\Phi \equiv \neg\Diamond^I\neg\Phi$ where $\mathsf{tt} := a \vee \neg a$ for some $a \in AP$. Similarly, the equality $\exists^{\sqsubseteq p}\varphi \equiv \neg\forall^{\sqsupset p}\varphi$ defines an existentially quantified transient state operator.

Example 3. Reconsider the CTMDP from Fig. 2(a). The *transient state formula* $\forall^{>0.1}\Diamond^{[0,1]}a$ states that the probability to reach an a-labelled state within at most one time unit exceeds 0.1 no matter how the nondeterministic choices in the current state are resolved. Further, the *long-run average formula* $\mathsf{L}^{<0.25}\neg a$ states that for all scheduling decisions, the system spends less than 25% of its execution time in non-a states, on average.

Formally the long-run average is derived as follows: For $B \subseteq \mathcal{S}$, let $\mathbf{I}_B$ denote an indicator with $\mathbf{I}_B(s) = 1$ if $s \in B$ and 0 otherwise. Following the ideas of [15,24], we compute the fraction of time spent in states from the set B on an infinite path π up to time bound $t \in \mathbb{R}_{\geq 0}$ and define $avg_{B,t}(\pi) = \frac{1}{t}\int_0^t \mathbf{I}_B(\pi@t')dt'$. As $avg_{B,t}$ is a random variable, its expectation can be derived given an initial distribution $\nu \in Distr(\mathcal{S})$ and a measurable scheduler $\mathcal{D} \in THR$, i.e. $E(avg_{B,t}) = \int_{Paths^\omega} avg_{B,t}(\pi)\ Pr^\omega_{\nu,\mathcal{D}}(d\pi)$. Having the expectation for fixed time bound t, we now let $t \to \infty$ and obtain the long-run average as $\lim_{t\to\infty} E(avg_{B,t})$.

Definition 9 (CSL semantics). *Let $\mathcal{C} = (\mathcal{S}, Act, \mathbf{R}, AP, L)$ be a CTMDP, $s, t \in \mathcal{S}$, $a \in AP$, $\sqsubseteq \in \{<, \leq, \geq, >\}$ and $\pi \in Paths^\omega$. Further let $\nu_s(t) := 1$ if $s = t$ and 0 otherwise. The semantics of* state formulas *is defined by*

$$\begin{aligned}
s &\models a \iff a \in L(s)\\
s &\models \neg\Phi \iff \textit{not } s \models \Phi\\
s &\models \Phi \wedge \Psi \iff s \models \Phi \textit{ and } s \models \Psi\\
s &\models \forall^{\sqsubseteq p}\varphi \iff \forall \mathcal{D} \in \mathit{THR}.\ \ \mathit{Pr}^{\omega}_{\nu_s,\mathcal{D}}\{\pi \in \mathit{Paths}^{\omega} \mid \pi \models \varphi\} \sqsubseteq p\\
s &\models \mathsf{L}^{\sqsubseteq p}\Phi \iff \forall \mathcal{D} \in \mathit{THR}.\ \lim_{t\to\infty}\int_{\mathit{Paths}^{\omega}} \mathit{avg}_{\mathit{Sat}(\Phi),t}(\pi)\ \ \mathit{Pr}^{\omega}_{\nu_s,\mathcal{D}}(d\pi)\ \sqsubseteq p.
\end{aligned}$$

Path formulas *are defined by*

$$\begin{aligned}
\pi &\models \mathsf{X}^I\Phi \iff \pi[1] \models \Phi \wedge \delta(\pi,0) \in I\\
\pi &\models \Phi\mathsf{U}^I\Psi \iff \exists t \in I.\ \big(\pi@t \models \Psi \wedge \big(\forall t' \in [0,t).\ \pi@t' \models \Phi\big)\big)
\end{aligned}$$

where $\mathit{Sat}(\Phi) := \{s \in \mathcal{S} \mid s \models \Phi\}$ and $\delta(\pi,n)$ is the time spent in state $\pi[n]$.

In Def. 9 the transient-state operator $\forall^{\sqsubseteq p}\varphi$ is based on the measure of the set of paths that satisfy φ. For this to be well defined we must show that the set $\{\pi \in \mathit{Paths}^{\omega} \mid \pi \models \varphi\}$ is measurable:

Theorem 2 (Measurability of path formulas). *For any* CSL *path formula* φ *the set* $\{\pi \in \mathit{Paths}^{\omega} \mid \pi \models \varphi\}$ *is measurable.*

Proof. For next formulas, the proof is straightforward. For until formulas, let $\pi = s_0 \xrightarrow{\alpha_0,t_0} s_1 \xrightarrow{\alpha_1,t_1} \cdots \in \mathit{Paths}^{\omega}$ and assume $\pi \models \Phi\mathsf{U}^I\Psi$. By Def. 9 it holds $\pi \models \Phi\mathsf{U}^I\Psi$ iff $\exists t \in I.\ \big(\pi@t \models \Psi \wedge \forall t' \in [0,t).\ \pi@t' \models \Phi\big)$. As we may exclude Zeno behaviour by Theorem 1, there exists $n \in \mathbb{N}$ with $\pi@t = \pi[n] = s_n$ such that I and the period of time $\big[\sum_{i=0}^{n-1} t_i, \sum_{i=0}^{n} t_i\big)$ spent in state s_n overlap; further $s_n \models \Psi$ and $s_i \models \Phi$ for $i = 0,\ldots,n-1$. Note however, that s_n must also satisfy Φ except for the case of *instantaneous arrival* where $\sum_{i=0}^{n-1} t_i \in I$. Accordingly, the set $\{\pi \in \mathit{Paths}^{\omega} \mid \pi \models \Phi\mathsf{U}^I\Psi\}$ can be represented by the union

$$\bigcup_{n=0}^{\infty}\Big\{\pi \in \mathit{Paths}^{\omega}\ \Big|\ \sum_{i=0}^{n-1} t_i \in I \wedge \pi[n] \models \Psi \wedge \forall m < n.\ \pi[m] \models \Phi\Big\} \tag{2}$$

$$\cup \bigcup_{n=0}^{\infty}\Big\{\pi \in \mathit{Paths}^{\omega}\ \Big|\ \big(\sum_{i=0}^{n-1} t_i, \sum_{i=0}^{n} t_i\big) \cap I \neq \emptyset \wedge \pi[n] \models \Psi \wedge \forall m \leq n.\ \pi[m] \models \Phi\Big\}. \tag{3}$$

It suffices to show that the subsets of (2) and (3) induced by any $n \in \mathbb{N}$ are measurable cylinders. In the following, we exhibit the proof for (3) and closed intervals $I = [a,b]$ as the other cases are similar. For fixed $n \geq 0$ we show that the corresponding cylinder base is measurable using a discretization argument:

$$\Big\{\pi \in \mathit{Paths}^{n+1}\ \Big|\ \big(\sum_{i=0}^{n-1} t_i, \sum_{i=0}^{n} t_i\big) \cap [a,b] \neq \emptyset \wedge \pi[n] \models \Psi \wedge \forall m \leq n.\ \pi[m] \models \Phi\Big\}$$

$$= \bigcup_{k=1}^{\infty}\ \bigcup_{\substack{c_0+\cdots+c_n \geq ak\\ d_0+\cdots+d_{n-1} \leq bk\\ c_i < d_i}}\ \prod_{i=0}^{n-1}\Big[\mathit{Sat}(\Phi) \times \mathit{Act} \times \Big(\frac{c_i}{k},\frac{d_i}{k}\Big)\Big] \times \mathit{Sat}(\Phi\wedge\Psi) \times \mathit{Act} \times \Big(\frac{c_n}{k},\infty\Big) \times \mathcal{S} \tag{4}$$

where $c_i, d_j \in \mathbb{N}$. To shorten notation, let $c := \sum_{i=0}^{n-1} t_i$ and $d := \sum_{i=0}^{n} t_i$.

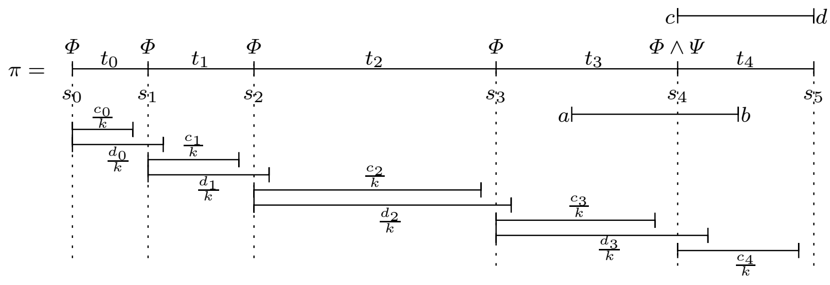


Fig. 3. Discretization of intervals with $n = 4$ and $I = (a, b)$

$\subseteq$: Let $\pi = s_0 \xrightarrow{\alpha_0, t_0} s_1 \xrightarrow{\alpha_1, t_1} \cdots \xrightarrow{\alpha_n, t_n} s_{n+1}$ be in the set on the left-hand side of equation (4). The intervals (c, d) and $[a, b]$ overlap, hence $c < b$ and $d > a$ (see top of Fig. 3). Further $\pi[i] \models \Phi$ for $i = 0, \ldots, n$ and $\pi[n] \models \Psi$. To show that π is in the set on the right-hand side, let $c_i = \lceil t_i \cdot k - 1 \rceil$ and $d_i = \lfloor t_i \cdot k + 1 \rfloor$ for $k > 0$. Then $\frac{c_i}{k} < t_i < \frac{d_i}{k}$ approximates the sojourn times t_i as depicted in Fig. 3. Further let $\varepsilon = \sum_{i=0}^{n} t_i - a$ and choose k_0 such that $\frac{n+1}{k_0} \leq \varepsilon$ to obtain

$$a = \sum_{i=0}^{n} t_i - \varepsilon \leq \sum_{i=0}^{n} t_i - \frac{n+1}{k_0} \leq \sum_{i=0}^{n} \frac{c_i + 1}{k_0} - \frac{n+1}{k_0} = \sum_{i=0}^{n} \frac{c_i}{k_0}.$$

Thus $ak \leq \sum_{i=0}^{n} c_i$ for all $k \geq k_0$. Similarly, we obtain $k_0' \in \mathbb{N}$ s.t. $\sum_{i=0}^{n-1} d_i \leq bk$ for all $k \geq k_0'$. Hence for large k, π is in the set on the right-hand side.

$\supseteq$: Let π be in the set on the right-hand side of equation (4) with corresponding values for c_i, d_i and k. Then $t_i \in \left(\frac{c_i}{k}, \frac{d_i}{k}\right)$. Hence $a \leq \sum_{i=0}^{n} \frac{c_i}{k} < \sum_{i=0}^{n} t_i = d$ and $b \geq \sum_{i=0}^{n-1} \frac{d_i}{k} > \sum_{i=0}^{n-1} t_i = c$ so that the time-interval (c, d) of state s_n and the time interval $I = [a, b]$ of the formula overlap. Further, $\pi[m] \models \Phi$ for $m \leq n$ and $\pi[n] \models \Psi$; thus π is in the set on the left-hand side of equation (4).

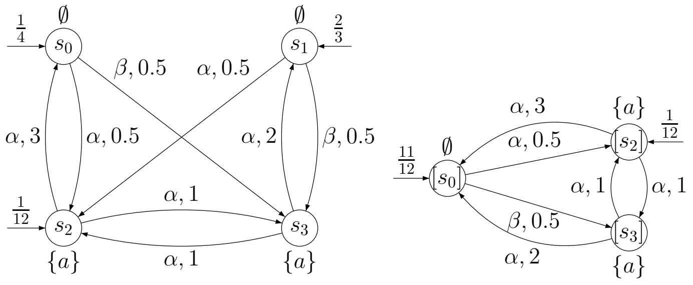
The right-hand side of equation (4) is measurable, hence also the cylinder base. This extends to its cylinder and the countable union in equation (3). $\square$

4.2 Strong Bisimilarity Preserves CSL

We now prepare the main result of our paper. To prove that strong bisimilarity preserves CSL formulas we establish a correspondence between certain sets of paths of a CTMDP and its quotient which is measure-preserving:

Definition 10 (Simple bisimulation closed). *Let $\mathcal{C} = (\mathcal{S}, Act, \mathbf{R}, AP, L)$ be a CTMDP. A measurable rectangle $\Pi = S_0 \times A_0 \times T_0 \times \cdots \times A_{n-1} \times T_{n-1} \times S_n$ is* simple bisimulation closed *if $S_i \in \left(\tilde{\mathcal{S}} \cup \{\emptyset\}\right)$ for $i = 0, \ldots, n$. Further, let $\tilde{\Pi} = \{S_0\} \times A_0 \times T_0 \times \cdots \times A_{n-1} \times T_{n-1} \times \{S_n\}$ be the corresponding rectangle in the quotient $\tilde{\mathcal{C}}$.*

An essential step in our proof strategy is to obtain a scheduler on the quotient. The following example illustrates the intuition for such a scheduler.



(a) CTMDP $\mathcal{C}$ and initial distr. (b) Quotient $\tilde{\mathcal{C}}$

Fig. 4. Derivation of the quotient scheduler

Example 4. Let $\mathcal{C}$ be the CTMDP in Fig. 4(a) where $\nu(s_0) = \frac{1}{4}$, $\nu(s_1) = \frac{2}{3}$ and $\nu(s_2) = \frac{1}{12}$. Assume a scheduler $\mathcal{D}$ where $\mathcal{D}(s_0, \{\alpha\}) = \frac{2}{3}$, $\mathcal{D}(s_0, \{\beta\}) = \frac{1}{3}$, $\mathcal{D}(s_1, \{\alpha\}) = \frac{1}{4}$ and $\mathcal{D}(s_1, \{\beta\}) = \frac{3}{4}$. Intuitively, a scheduler $\mathcal{D}^{\nu}_{\sim}$ that mimics $\mathcal{D}$'s behaviour on the quotient $\tilde{\mathcal{C}}$ in Fig. 4(b) can be defined by

$$\mathcal{D}^{\nu}_{\sim}([s_0], \{\alpha\}) = \frac{\sum_{s\in[s_0]} \nu(s) \cdot \mathcal{D}(s, \{\alpha\})}{\sum_{s\in[s_0]} \nu(s)} = \frac{\frac{1}{4}\cdot\frac{2}{3} + \frac{2}{3}\cdot\frac{1}{4}}{\frac{1}{4}+\frac{2}{3}} = \frac{4}{11} \quad \text{and}$$

$$\mathcal{D}^{\nu}_{\sim}([s_0], \{\beta\}) = \frac{\sum_{s\in[s_0]} \nu(s) \cdot \mathcal{D}(s, \{\beta\})}{\sum_{s\in[s_0]} \nu(s)} = \frac{\frac{1}{4}\cdot\frac{1}{3} + \frac{2}{3}\cdot\frac{3}{4}}{\frac{1}{4}+\frac{2}{3}} = \frac{7}{11}.$$

Even though s_0 and s_1 are bisimilar, the scheduler $\mathcal{D}$ decides differently for the histories $\pi_0 = s_0$ and $\pi_1 = s_1$. As π_0 and π_1 collapse into $\tilde{\pi} = [s_0]$ on the quotient, $\mathcal{D}^{\nu}_{\sim}$ can no longer distinguish between π_0 and π_1. Therefore $\mathcal{D}$'s decision for any history $\pi \in \tilde{\pi}$ is weighed w.r.t. the total probability of $\tilde{\pi}$.

Definition 11 (Quotient scheduler). *Let $\mathcal{C} = (\mathcal{S}, Act, \mathbf{R}, AP, L)$ be a CTMDP, $\nu \in Distr(\mathcal{S})$ and $\mathcal{D} \in THR$. First, define the* history weight *of finite paths of length n inductively as follows:*

$$hw_0(\nu, \mathcal{D}, s_0) := \nu(s_0) \ \textit{and}$$

$$hw_{n+1}(\nu, \mathcal{D}, \pi \xrightarrow{\alpha_n, t_n} s_{n+1}) := hw_n(\nu, \mathcal{D}, \pi) \cdot \mathcal{D}(\pi, \{\alpha_n\}) \cdot \mathbf{P}(\pi{\downarrow}, \alpha_n, s_{n+1}).$$

Let $\tilde{\pi} = [s_0] \xrightarrow{\alpha_0, t_0} \cdots \xrightarrow{\alpha_{n-1}, t_{n-1}} [s_n]$ be a timed history of $\tilde{\mathcal{C}}$ and $\Pi = [s_0] \times \{\alpha_0\} \times \{t_0\} \times \cdots \times \{\alpha_{n-1}\} \times \{t_{n-1}\} \times [s_n]$ be the corresponding set of paths in $\mathcal{C}$. The quotient scheduler *$\mathcal{D}^{\nu}_{\sim}$ on $\tilde{\mathcal{C}}$ is then defined as follows:*

$$\mathcal{D}^{\nu}_{\sim}(\tilde{\pi}, \alpha_n) := \frac{\sum_{\pi\in\Pi} hw_n(\nu, \mathcal{D}, \pi) \cdot \mathcal{D}(\pi, \{\alpha_n\})}{\sum_{\pi\in\Pi} hw_n(\nu, \mathcal{D}, \pi)}.$$

Further, let $\tilde{\nu}([s]) := \sum_{s'\in[s]} \nu(s')$ be the initial distribution on $\tilde{\mathcal{C}}$.

A history $\tilde{\pi}$ of $\tilde{\mathcal{C}}$ corresponds to a set of paths Π in $\mathcal{C}$; given $\tilde{\pi}$, the quotient scheduler decides by multiplying $\mathcal{D}$'s decision on each path in Π with its corresponding weight and normalizing with the weight of Π afterwards. Now we obtain a first intermediate result: For CTMDP $\mathcal{C}$, if Π is a simple bisimulation closed set of paths, ν an initial distribution and $\mathcal{D} \in \mathit{THR}$, the measure of Π in $\mathcal{C}$ coincides with the measure of $\tilde{\Pi}$ in $\tilde{\mathcal{C}}$ which is induced by $\tilde{\nu}$ and $\mathcal{D}^{\nu}_{\sim}$:

Theorem 3. *Let $\mathcal{C}$ be a CTMDP with set of states $\mathcal{S}$ and $\nu \in \mathit{Distr}(\mathcal{S})$. Then $Pr^{\omega}_{\nu,\mathcal{D}}(\Pi) = Pr^{\omega}_{\tilde{\nu},\mathcal{D}^{\nu}_{\sim}}(\tilde{\Pi})$ where $\mathcal{D} \in \mathit{THR}$ and Π simple bisimulation closed.*

Proof. By induction on the length n of cylinder bases. The induction base holds for $Pr^{0}_{\nu,\mathcal{D}}\big([s]\big) = \sum_{s' \in [s]} \nu(s') = \tilde{\nu}\big([s]\big) = Pr^{0}_{\tilde{\nu},\mathcal{D}^{\nu}_{\sim}}\big(\{[s]\}\big)$. With the induction hypothesis that $Pr^{n}_{\nu,\mathcal{D}}(\Pi) = Pr^{n}_{\tilde{\nu},\mathcal{D}^{\nu}_{\sim}}(\tilde{\Pi})$ for all $\nu \in \mathit{Distr}(\mathcal{S})$, $\mathcal{D} \in \mathit{THR}$ and bisimulation closed $\Pi \subseteq \mathit{Paths}^n$ we obtain the induction step:

$$\begin{aligned}
Pr^{n+1}_{\nu,\mathcal{D}}\big([s_0] \times A_0 \times T_0 \times \Pi\big) &= \int_{[s_0]\times A_0 \times T_0} Pr^{n}_{\mathbf{P}(s,\alpha,\cdot),\mathcal{D}(s\xrightarrow{\alpha,t}\cdot)}(\Pi)\ \mu_{\nu,\mathcal{D}}(ds, d\alpha, dt)\\
&= \int_{s\in[s_0]} \nu(ds) \int_{\alpha \in A_0} \mathcal{D}(s, d\alpha) \int_{T_0} Pr^{n}_{\mathbf{P}(s,\alpha,\cdot),\mathcal{D}(s\xrightarrow{\alpha,t}\cdot)}(\Pi)\ \eta_{E(s,\alpha)}(dt)\\
&= \sum_{s\in[s_0]} \nu(s) \sum_{\alpha\in A_0} \mathcal{D}(s,\{\alpha\}) \int_{T_0} Pr^{n}_{\mathbf{P}(s,\alpha,\cdot),\mathcal{D}(s\xrightarrow{\alpha,t}\cdot)}(\Pi)\ \eta_{\tilde{E}([s_0],\alpha)}(dt)\\
&\overset{\text{i.h.}}{=} \sum_{s\in[s_0]} \sum_{\alpha\in A_0} \int_{T_0} Pr^{n}_{\tilde{\mathbf{P}}([s_0],\alpha,\cdot),\mathcal{D}^{\nu}_{\sim}([s_0]\xrightarrow{\alpha,t}\cdot)}(\tilde{\Pi}) \cdot \nu(s) \cdot \mathcal{D}(s,\{\alpha\})\ \eta_{\tilde{E}([s_0],\alpha)}(dt)\\
&= \sum_{\alpha\in A_0} \int_{T_0} Pr^{n}_{\tilde{\mathbf{P}}([s_0],\alpha,\cdot),\mathcal{D}^{\nu}_{\sim}([s_0]\xrightarrow{\alpha,t}\cdot)}(\tilde{\Pi}) \cdot \sum_{s\in[s_0]} \Big(\nu(s) \cdot \mathcal{D}(s,\{\alpha\})\Big)\ \eta_{\tilde{E}([s_0],\alpha)}(dt)\\
&= \sum_{\alpha\in A_0} \int_{T_0} Pr^{n}_{\tilde{\mathbf{P}}([s_0],\alpha,\cdot),\mathcal{D}^{\nu}_{\sim}([s_0]\xrightarrow{\alpha,t}\cdot)}(\tilde{\Pi}) \cdot \tilde{\nu}([s_0]) \cdot \mathcal{D}^{\nu}_{\sim}([s_0],\{\alpha\})\ \eta_{\tilde{E}([s_0],\alpha)}(dt)\\
&= \int_{\{[s_0]\}} \tilde{\nu}(d\,[s]) \int_{A_0} \mathcal{D}^{\nu}_{\sim}([s], d\alpha) \int_{T_0} Pr^{n}_{\tilde{\mathbf{P}}([s],\alpha,\cdot),\mathcal{D}^{\nu}_{\sim}([s]\xrightarrow{\alpha,t}\cdot)}(\tilde{\Pi})\ \eta_{\tilde{E}([s],\alpha)}(dt)\\
&= \int_{\{[s_0]\}\times A_0 \times T_0} Pr^{n}_{\tilde{\mathbf{P}}([s],\alpha,\cdot),\mathcal{D}^{\nu}_{\sim}([s]\xrightarrow{\alpha,t}\cdot)}(\tilde{\Pi})\ \tilde{\mu}_{\tilde{\nu},\mathcal{D}^{\nu}_{\sim}}(d\,[s], d\alpha, dt)\\
&= Pr^{n+1}_{\tilde{\nu},\mathcal{D}^{\nu}_{\sim}}\big(\{[s_0]\} \times A_0 \times T_0 \times \tilde{\Pi}\big)
\end{aligned}$$

where $\tilde{\mu}_{\tilde{\nu},\mathcal{D}^{\nu}_{\sim}}$ is the extension of $\mu_{\nu,\mathcal{D}}$ (Def. 5) to sets of initial triples in $\tilde{\mathcal{C}}$:

$$\tilde{\mu}_{\tilde{\nu},\mathcal{D}^{\nu}_{\sim}} \colon \mathfrak{F}_{\tilde{\mathcal{S}}\times \mathit{Act} \times \mathbb{R}_{\geq 0}} \to [0,1] : I \mapsto \int_{\tilde{\mathcal{S}}} \tilde{\nu}(d\,[s]) \int_{\mathit{Act}} \mathcal{D}^{\nu}_{\sim}([s], d\alpha) \int_{\mathbb{R}_{\geq 0}} \mathbf{I}_I([s],\alpha,t)\ \eta_{\tilde{E}([s],\alpha)}(dt). \quad \square$$

According to Theorem 3, the quotient scheduler preserves the measure for *simple* bisimulation closed sets of paths, i.e. for paths, whose state components are equivalence classes under $\sim$. To generalize this to sets of paths that satisfy a CSL path formula, we introduce *general* bisimulation closed sets of paths:

Definition 12 (Bisimulation closed). *Let $\mathcal{C} = (\mathcal{S}, \mathit{Act}, \mathbf{R}, AP, L)$ be a CTMDP and $\tilde{\mathcal{C}}$ its quotient under strong bisimilarity. A measurable rectangle $\Pi = S_0 \times A_0 \times T_0 \times \cdots \times A_{n-1} \times T_{n-1} \times S_n$ is* bisimulation closed *if $S_i = \biguplus_{j=0}^{k_i} [s_{i,j}]$ for $k_i \in \mathbb{N}$ and $0 \leq i \leq n$. Let $\tilde{\Pi} = \bigcup_{j=0}^{k_0} \big\{[s_{0,j}]\big\} \times A_0 \times T_0 \times \cdots \times A_{n-1} \times T_{n-1} \times \bigcup_{j=0}^{k_n} \big\{[s_{n,j}]\big\}$ be the corresponding rectangle in the quotient $\tilde{\mathcal{C}}$.*

Lemma 2. *Any bisimulation closed set of paths Π can be represented as a finite disjoint union of* simple *bisimulation closed sets of paths.*

Proof. Direct consequence of Def. 12. □

Corollary 1. *Let $\mathcal{C}$ be a CTMDP with set of states $\mathcal{S}$ and $\nu \in Distr(\mathcal{S})$ an initial distribution. Then $Pr^{\omega}_{\nu,\mathcal{D}}(\Pi) = Pr^{\omega}_{\tilde{\nu},\mathcal{D}^{\nu}_{\sim}}(\tilde{\Pi})$ for any $\mathcal{D} \in THR$ and any* bisimulation closed *set of paths Π.*

Proof. Follows directly from Lemma 2 and Theorem 3. □

Using these extensions we can now prove our main result:

Theorem 4. *Let $\mathcal{C}$ be a CTMDP with set of states $\mathcal{S}$ and $u, v \in \mathcal{S}$. Then $u \sim v$ implies $u \models \Phi$ iff $v \models \Phi$ for all* CSL *state formulas Φ.*

Proof. By structural induction on Φ. If $\Phi = a$ and $a \in AP$ the induction base follows as $L(u) = L(v)$. In the induction step, conjunction and negation are obvious.

Let $\Phi = \forall^{\sqsubseteq p}\varphi$ and $\Pi = \{\pi \in Paths^{\omega} \mid \pi \models \varphi\}$. To show $u \models \forall^{\sqsubseteq p}\varphi$ implies $v \models \forall^{\sqsubseteq p}\varphi$ it suffices to show that for any $\mathcal{V} \in THR$ there exists $\mathcal{U} \in THR$ with $Pr^{\omega}_{\nu_u,\mathcal{U}}(\Pi) = Pr^{\omega}_{\nu_v,\mathcal{V}}(\Pi)$. By Theorem 2 the set Π is measurable, hence $\Pi = \biguplus_{i=0}^{\infty} \Pi_i$ for disjoint $\Pi_i \in \mathfrak{F}_{Paths^{\omega}}$. By *induction hypothesis* for path formulas $\mathsf{X}^I\Phi$ and $\Phi\mathsf{U}^I\Psi$ the sets $Sat(\Phi)$ and $Sat(\Psi)$ are disjoint unions of $\sim$-equivalence classes. The same holds for any Boolean combination of Φ and Ψ. Hence $\Pi = \biguplus_{i=0}^{\infty} \Pi_i$ where the Π_i are bisimulation closed. For all $\mathcal{V} \in THR$ and $\pi = s_0 \xrightarrow{\alpha_0,t_0} \cdots \xrightarrow{\alpha_{n-1},t_{n-1}} s_n$ let $\mathcal{U}(\pi) := \mathcal{V}^{\nu_v}_{\sim}\big([s_0] \xrightarrow{\alpha_0,t_0} \cdots \xrightarrow{\alpha_{n-1},t_{n-1}} [s_n]\big)$. Thus $\mathcal{U}$ mimics on π the decision of $\mathcal{V}^{\nu_v}_{\sim}$ on $\tilde{\pi}$. In fact $\mathcal{U}^{\nu_u}_{\sim} = \mathcal{V}^{\nu_v}_{\sim}$ since

$$\mathcal{U}^{\nu_u}_{\sim}(\tilde{\pi}, \alpha_n) = \frac{\sum_{\pi \in \Pi} hw_n(\nu_u, \mathcal{U}, \pi) \cdot \mathcal{V}^{\nu_v}_{\sim}(\tilde{\pi}, \alpha_n)}{\sum_{\pi \in \Pi} hw_n(\nu_u, \mathcal{U}, \pi)}$$

and $\mathcal{V}^{\nu_v}_{\sim}(\tilde{\pi}, \alpha_n)$ is independent of π. With $\tilde{\nu}_u = \tilde{\nu}_v$ and by Corollary 1 we obtain $Pr^{\omega}_{\nu_u,\mathcal{U}}(\Pi_i) = Pr^{\omega}_{\tilde{\nu}_u,\mathcal{U}^{\nu_u}_{\sim}}(\tilde{\Pi}_i) = Pr^{\omega}_{\tilde{\nu}_v,\mathcal{V}^{\nu_v}_{\sim}}(\tilde{\Pi}_i) = Pr^{\omega}_{\nu_v,\mathcal{V}}(\Pi_i)$ which carries over to Π for Π is a countable union of disjoint sets Π_i.

Let $\Phi = \mathsf{L}^{\sqsubseteq p}\Psi$. Since $u \sim v$, it suffices to show that for all $s \in \mathcal{S}$ it holds $s \models \mathsf{L}^{\sqsubseteq p}\Psi$ iff $[s] \models \mathsf{L}^{\sqsubseteq p}\Psi$. The expectation of $avg_{Sat(\Psi),t}$ for $t \in \mathbb{R}_{\geq 0}$ can be expressed as follows:

$$\int_{Paths^{\omega}} \left(\frac{1}{t}\int_0^t \mathbf{I}_{Sat(\Psi)}(\pi@t')dt'\right) Pr^{\omega}_{\nu_s,\mathcal{D}}(d\pi) = \frac{1}{t}\int_0^t Pr^{\omega}_{\nu_s,\mathcal{D}}\{\pi \in Paths^{\omega} \mid \pi@t' \models \Psi\}dt'.$$

Further, the sets $\{\pi \in Paths^{\omega} \mid \pi@t' \models \Psi\}$ and $\{\pi \in Paths^{\omega} \mid \pi \models \Diamond^{[t',t']}\Psi\}$ have the same measure and the *induction hypothesis* applies to Ψ. Applying the previous reasoning for the until case to the formula $\mathsf{tt}\,\mathsf{U}^{[t',t']}\Psi$ once, we obtain

$$Pr^{\omega}_{\nu_s,\mathcal{D}}\{\pi \in Paths^{\omega}(\mathcal{C}) \mid \pi \models \Diamond^{[t',t']}\Psi\} = Pr^{\omega}_{\tilde{\nu}_s,\mathcal{D}^{\nu_s}_{\sim}}\{\tilde{\pi} \in Paths^{\omega}(\tilde{\mathcal{C}}) \mid \tilde{\pi} \models \Diamond^{[t',t']}\Psi\}$$

for all $t' \in \mathbb{R}_{\geq 0}$. Thus the expectations of $avg_{Sat(\Psi),t}$ on $\mathcal{C}$ and $\tilde{\mathcal{C}}$ are equal for all $t \in \mathbb{R}_{\geq 0}$ and the same holds for their limits if $t \to \infty$. This completes the proof as for $u \sim v$ we obtain $u \models \mathsf{L}^{\sqsubseteq p}\Psi$ iff $[u] \models \mathsf{L}^{\sqsubseteq p}\Psi$ iff $[v] \models \mathsf{L}^{\sqsubseteq p}\Psi$ iff $v \models \mathsf{L}^{\sqsubseteq p}\Psi$. □

This theorem shows that bisimilar states satisfy the same CSL formulas. The reverse direction, however, does not hold in general. One reason is obvious: In this paper we use a purely state-based logic whereas our definition of strong bisimulation also accounts for action names. Therefore it comes to no surprise that CSL cannot characterize strong bisimulation. However, there is another more profound reason which is analogous to the discrete-time setting where extensions of PCTL to Markov decision processes [29,4] also cannot express strong bisimilarity: CSL and PCTL only allow to specify infima and suprema as probability bounds under a denumerable class of randomized schedulers; therefore intuitively, CSL cannot characterize exponential distributions which neither contribute to the supremum nor to the infimum of the probability measures of a given set of paths. Thus the counterexample from [4, Fig 9.5] interpreted as a CTMDP applies verbatim to our case.

5 Conclusion

In this paper we define strong bisimulation on CTMDPs and propose a nondeterministic extension of CSL to CTMDP that allows to express a wide class of performance and dependability measures. Using a measure-theoretic argument we prove our logic to be well-defined. Our main contribution is the proof that strong bisimilarity preserves the validity of CSL formulas. However, our logic is not capable of characterizing strong bisimilarity. To this end, action-based logics provide a natural starting point.

Acknowledgements. This research has been performed as part of the QUPES project that is financed by the Netherlands Organization for Scientific Research (NWO). Daniel Klink and David N. Jansen are kindly acknowledged for many fruitful discussions.

References

1. Abdeddaïm, Y., Asarin, E., Maler, O.: On optimal scheduling under uncertainty. In: Garavel, H., Hatcliff, J. (eds.) ETAPS 2003 and TACAS 2003. LNCS, vol. 2619, pp. 240–253. Springer, Heidelberg (2003)
2. Ash, R.B., Doléans-Dade, C.A.: Probability & Measure Theory, 2nd edn. Academic Press, London (2000)
3. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.K.: Model-checking continous-time Markov chains. ACM Trans. Comput. Log. 1, 162–170 (2000)
4. Baier, C.: On Algorithmic Verification Methods for Probabilistic Systems. Habilitation Thesis, University of Mannheim (1998)
5. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P.: Model-checking algorithms for continuous-time Markov chains. IEEE TSE 29, 524–541 (2003)
6. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P.: Nonuniform CTMDPs. unpublished manuscript (2004)

7. Baier, C., Hermanns, H., Katoen, J.-P., Haverkort, B.R.: Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. Theor. Comp. Sci. 345, 2–26 (2005)
8. Baier, C., Katoen, J.-P., Hermanns, H., Wolf, V.: Comparative branching-time semantics for Markov chains. Information and Computation 200, 149–214 (2005)
9. Baier, C., Kwiatkowska, M.Z.: Model checking for a probabilistic branching time logic with fairness. Distr. Comp. 11, 125–155 (1998)
10. Beutler, F.J., Ross, K.W.: Optimal policies for controlled Markov chains with a constraint. Journal of Mathematical Analysis and Appl. 112, 236–252 (1985)
11. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed.) Foundations of Software Technology and Theoretical Computer Science. LNCS, vol. 1026, pp. 499–513. Springer, Heidelberg (1995)
12. Boudali, H., Crouzen, P., Stoelinga, M.I.A.: Dynamic fault tree analysis using input/output interactive Markov chains. In: Dependable Systems and Networks, IEEE Computer Society Press, Los Alamitos (2007)
13. Buchholz, P.: Exact and ordinary lumpability in finite Markov chains. Journal of Applied Probability 31, 59–75 (1994)
14. Chiola, G., Marsan, M.A., Balbo, G., Conte, G.: Generalized stochastic Petri nets: A definition at the net level and its implications. IEEE TSE 19, 89–107 (1993)
15. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD thesis, Stanford University (1997)
16. Desharnais, J., Panangaden, P.: Continuous stochastic logic characterizes bisimulation of continuous-time Markov processes. Journal of Logic and Algebraic Programming 56, 99–115 (2003)
17. Feinberg, E.A.: Continuous time discounted jump Markov decision processes: A discrete-event approach. Mathematics of Operations Research 29, 492–524 (2004)
18. Givan, R., Dean, T., Greig, M.: Equivalence notions and model minimization in Markov decision processes. Artificial Intelligence 147, 163–223 (2003)
19. Hermanns, H.: Interactive Markov Chains. LNCS, vol. 2428. Springer, Heidelberg (2002)
20. Howard, R.A.: Dynamic Probabilistic Systems. John Wiley and Sons, West Sussex, England (1971)
21. Katoen, J.-P., Klink, D., Leucker, M., Wolf, V.: Three-valued abstraction for continuous-time Markov chains. In: CAV. LNCS, Springer, Heidelberg (2007)
22. Kemeny, J.G., Snell, J.L., Knapp, A.W.: Denumerable Markov Chains, 2nd edn. Springer, Heidelberg (1976)
23. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Information and Computation 94, 1–28 (1991)
24. López, G.G.I., Hermanns, H., Katoen, J.-P.: Beyond memoryless distributions: Model checking semi-Markov chains. In: de Luca, L., Gilmore, S.T. (eds.) PROBMIV 2001, PAPM-PROBMIV 2001, and PAPM 2001. LNCS, vol. 2165, pp. 57–70. Springer, Heidelberg (2001)
25. Neuhäußer, M.R.: Bisimulation and logical preservation for continuous-time markov decision processes. Technical Report 10, RWTH Aachen (2007)
26. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley and Sons, West Sussex, England (1994)

27. Qiu, Q., Pedram, M.: Dynamic power management based on continuous-time Markov decision processes. In: DAC, pp. 555–561. ACM Press, New York (1999)
28. Sanders, W.H., Meyer, J.F.: Stochastic activity networks: Formal definitions and concepts. In: Brinksma, E., Hermanns, H., Katoen, J.-P. (eds.) Lectures on Formal Methods and Performance Analysis. LNCS, vol. 2090, pp. 315–343. Springer, Heidelberg (2001)
29. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing 2, 250–273 (1995)
30. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: FOCS, pp. 327–338. IEEE Computer Society Press, Los Alamitos (1985)
31. Wolovick, N., Johr, S.: A characterization of meaningful schedulers for continuous-time Markov decision processes. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 352–367. Springer, Heidelberg (2006)

Strategy Synthesis for Markov Decision Processes and Branching-Time Logics

Tomáš Brázdil* and Vojtěch Forejt**

Faculty of Informatics, Masaryk University,
Botanická 68a, 60200 Brno,
Czech Republic
{brazdil,forejt}@fi.muni.cz

Abstract. We consider a class of finite $1\frac{1}{2}$-player games (Markov decision processes) where the winning objectives are specified in the branching-time temporal logic qPECTL* (an extension of the qualitative PCTL*). We study decidability and complexity of existence of a winning strategy in these games. We identify a fragment of qPECTL* called detPECTL* for which the existence of a winning strategy is decidable in exponential time, and also the winning strategy can be computed in exponential time (if it exists). Consequently we show that every formula of qPECTL* can be translated to a formula of detPECTL* (in exponential time) so that the resulting formula is equivalent to the original one over finite Markov chains. From this we obtain that for the whole qPECTL*, the existence of a winning finite-memory strategy is decidable in double exponential time. An immediate consequence is that the existence of a winning finite-memory strategy is decidable for the qualitative fragment of PCTL* in triple exponential time. We also obtain a single exponential upper bound on the same problem for the qualitative PCTL. Finally, we study the power of finite-memory strategies with respect to objectives described in the qualitative PCTL.

1 Introduction

We study $1\frac{1}{2}$-player games (Markov decision processes), which have been applied in various contexts, from computer science and engineering (models of network systems, models of industrial processes, etc.) to biology [13,9,12]. A $1\frac{1}{2}$-player game G is a directed graph whose vertices are partitioned into two disjoint sets $V_\Box$ and $V_\bigcirc$. For each vertex of $V_\bigcirc$ there is a fixed probability distribution on outgoing transitions. A *play* is initiated by putting a token on some vertex. This token is then moved from vertex to vertex by one 'real' player $\Box$ and one 'virtual' player $\bigcirc$, who choose their moves in vertices of $V_\Box$ and $V_\bigcirc$, respectively. Player $\bigcirc$ chooses his moves randomly according to the fixed distribution. Player $\Box$ chooses his moves according to a *strategy*. Generally, strategies may depend on history of the play and may be either randomized or deterministic (we denote HR and HD the classes of the history-dependent randomized and deterministic strategies, respectively). In this paper we also consider *finite-memory*

* Supported by "Institute for Theoretical Computer Science (ITI)", project No. 1M0545.
** Supported by the Czech Science Foundation, project No. 102/05/H050.

strategies that depend on a finite-state information about the history of the play[1]. The classes of randomized and deterministic finite-memory strategies are denoted FR and FD, respectively.

Once player $\Box$ fixes his strategy σ for the game G, we obtain a Markov chain $G(\sigma)$ where the states are finite paths in G, and $ws \xrightarrow{x} wst$ if and only if (s,t) is a transition in G and x is either the fixed probability assigned to (s,t) (if $s \in V_{\bigcirc}$), or the probability of (s,t) assigned by player $\Box$ in ws. Now we may ask whether the resulting Markov chain $G(\sigma)$ satisfies a given property. A *winning objective* is a property of Markov chains to be achieved by player $\Box$. A strategy σ is called *winning* if the Markov chain $G(\sigma)$ satisfies the winning objective.

Winning objectives can be expressed using various formalisms. For example, various kinds of linear-time objectives, such as Büchi, parity, and Rabin objectives, were intensively studied in the past (see, e.g., [7,8,6]). In this paper we concentrate on a different kind of winning objectives specified by formulae of a branching-time temporal logic.

Let us note that the semantics of branching-time formulae can be defined directly for $1\frac{1}{2}$-player games (see, e.g., [2]). In that case strategies are chosen separately for each temporal operator occurring in a formula. This approach is different from the one taken in this paper and results on model-checking such games are not related to our results.

The problem of solving games with branching-time winning objectives was for the first time studied in [11], where the existence of a winning memoryless randomized strategy for objectives expressed in PCTL (see, e.g., [10]) was shown to be in **PSPACE**. Results of [11] were substantially extended in [3] where also history dependent strategies were taken into account. The most relevant results of [3] are the following. First, the existence of a winning HD (and also HR, FR and FD) strategy is undecidable for $1\frac{1}{2}$-player games with objectives specified in (quantitative) PCTL. Second, the problem of existence of a winning HD strategy is **EXPTIME**-complete for $1\frac{1}{2}$-player games with objectives specified in the $\mathcal{L}(\mathrm{F}^{=1}, \mathrm{G}^{=1}, \mathrm{F}^{>0})$[2] fragment of the qualitative PCTL.

The question is whether the positive result about the fragment $\mathcal{L}(\mathrm{F}^{=1}, \mathrm{G}^{=1}, \mathrm{F}^{>0})$ can be extended to more expressive logics at least for finite-memory strategies. In this paper we address this problem and show that the existence of a winning finite-memory strategy is decidable even for a powerful temporal logic qPECTL*. We also show that the winning finite-memory strategy can always be effectively synthesized. This problem is well motivated because in practice one usually does not only want to know whether a strategy exists but also wants to implement the strategy. Finite-memory strategies have the advantage of being easy to implement.

The logic qPECTL* is the qualitative fragment of the logic PECTL* defined in [5]. PECTL* is a generalization of the logic PCTL* (see, e.g., [5,2]) which is a probabilistic version of the well-known logic CTL*. Of course, PECTL* contains the logic PCTL. Hence, our results on qPECTL* have immediate consequences for the *qualitative* PCTL* (denoted qPCTL*) and the *qualitative* PCTL (denoted qPCTL).

[1] More formally, a finite-memory (randomized) strategy is represented by a deterministic finite-state automaton and a function which assigns a distribution on outgoing transitions to the current vertex of the play and the state of the automaton after reading the history of the play.

[2] Formulae of $\mathcal{L}(\mathrm{F}^{=1}, \mathrm{G}^{=1}, \mathrm{F}^{>0})$ are built up from literals using conjunction, disjunction, and the temporal operators $\mathrm{F}^{=1}, \mathrm{G}^{=1}, \mathrm{F}^{>0}$ (negation is applied only to atomic propositions).

Our contribution: The main results of this paper are summarized below.

- We show that the existence of a winning FR (or FD) strategy for objectives described by qPCTL, qPECTL*, and qPCTL* formulae is decidable in single exponential, double exponential, and triple exponential time, respectively. We also show that the winning strategy can effectively be computed with the same complexity. Moreover, we show that all these problems can be solved in time polynomial in the size of games.
- In the course of the proof of the above results we identify a fragment of qPECTL*, called detPECTL*, and show that the existence of a winning HR (or HD) strategy for objectives described in detPECTL* is decidable in time exponential in the size of formulae and polynomial in the size of games. The fragment detPECTL* contains the logic $\mathcal{L}(\mathrm{F}^{=1}, \mathrm{G}^{=1}, \mathrm{F}^{>0})$, and hence our results improve on the corresponding results of [3] by considering a more general logic, randomized strategies, and providing a polynomial time upper bound in the size of games.
- Finally, it has been shown in [3] that an infinite-memory strategy is needed for satisfying a formula of the fragment $\mathcal{L}(\mathrm{G}^{>0}, \mathrm{F}^{>0})$ of qPCTL. We extend this result and provide (in a sense) complete classification of the power of finite-memory strategies for various fragments of qPCTL.

Plan of the paper: In Section 2 we review basic definitions for Markov chains and games. We also introduce the logic qPECTL* and its fragments. In Section 3 we consider the problem of existence of a winning history-dependent strategy for objectives described in detPECTL*. In Section 4 we consider the same problem for finite-memory strategies and qPECTL*. Finally, Section 5 deals with the classification of fragments of qPCTL with respect to the power of finite-memory strategies.

2 Basic Definitions

In this section we introduce basic notions of Markov chains, probabilistic temporal logics, and games. Most definitions (except the definition of qPECTL*) are taken from [3].

We start by recalling basic notions of probability theory. Let A be a finite set. A *probability distribution* on A is a function $f : A \to [0, 1]$ such that $\sum_{a \in A} f(a) = 1$. A distribution f is *Dirac* if $f(a) = 1$ for some $a \in A$. The set of all distributions on A is denoted $\mathcal{D}(A)$.

A *σ-field* over a set X is a set $\mathcal{F} \subseteq 2^X$ that includes X and is closed under complement and countable union. A *measurable space* is a pair $(X, \mathcal{F})$ where X is a set called *sample space* and $\mathcal{F}$ is a σ-field over X. A *probability measure* over a measurable space $(X, \mathcal{F})$ is a function $\mathcal{P} : \mathcal{F} \to \mathbb{R}^{\geq 0}$ such that, for each countable collection $\{X_i\}_{i \in I}$ of pairwise disjoint elements of $\mathcal{F}$, $\mathcal{P}(\bigcup_{i \in I} X_i) = \sum_{i \in I} \mathcal{P}(X_i)$, and moreover $\mathcal{P}(X)$=1. A *probability space* is a triple $(X, \mathcal{F}, \mathcal{P})$ where $(X, \mathcal{F})$ is a measurable space and $\mathcal{P}$ is a probability measure over $(X, \mathcal{F})$.

2.1 Markov Chains

A *Markov chain* is a triple $M = (S, \rightarrow, \mathit{Prob})$ where S is a finite or countably infinite set of *states*, $\rightarrow \subseteq S \times S$ is a *transition relation*, and Prob is a function which to each transition $s \rightarrow t$ of M assigns its probability $\mathit{Prob}(s \rightarrow t) \in (0,1]$ so that for every $s \in S$ we have $\sum_{s \rightarrow t} \mathit{Prob}(s \rightarrow t) = 1$.

In the rest of this paper we also write $s \xrightarrow{x} t$ instead of $\mathit{Prob}(s \rightarrow t) = x$. A *path* in M is a finite or infinite sequence $w = s_0, s_1, \ldots$ of states such that $s_i \rightarrow s_{i+1}$ for every i. The *length* of a given path w is the number of transitions in w. We also use $w(i)$ to denote the state s_i of w (by writing $w(i) = s$ we implicitly impose the condition that the length of w is at least i). The prefix $s_0, s_1, \ldots, s_i$ of w is denoted by w^i. A *run* is an infinite path. The sets of all finite paths and all runs of M are denoted FPath and Run, respectively. Similarly, the sets of all finite paths and runs that start in a given $s \in S$ are denoted $\mathit{FPath}(s)$ and $\mathit{Run}(s)$, respectively.

We say that a set $C \subseteq S$ is a bottom strongly connected component (BSCC) of M if for all $s, t \in C$ there is a path from s to t in M, and whenever there is a path from $s \in C$ to $t \in S$, then $t \in C$. Note that if we restrict the set of states of M to a BSCC C, we obtain a Markov chain.

Each $w \in \mathit{FPath}$ determines a *basic cylinder* $\mathit{Run}(w)$ which consists of all runs that start with w. To every $s \in S$ we associate the probability space $(\mathit{Run}(s), \mathcal{F}, \mathcal{P})$ where $\mathcal{F}$ is the σ-field generated by all basic cylinders $\mathit{Run}(w)$ where w starts with s, and $\mathcal{P} : \mathcal{F} \rightarrow [0,1]$ is the unique probability measure such that $\mathcal{P}(\mathit{Run}(w)) = \Pi_{i=0}^{m-1} x_i$ where $w = s_0, \cdots, s_m$ and $s_i \xrightarrow{x_i} s_{i+1}$ for every $0 \leq i < m$ (if m=0, we put $\mathcal{P}(\mathit{Run}(w)) = 1$).

2.2 The Logic qPECTL*

A Büchi automaton is a tuple $\mathcal{B} = (B, \Sigma, \delta, q_I, F)$, where Σ is a finite *alphabet*, B is a finite set of *states*, $\delta \subseteq B \times \Sigma \times B$ is a *transition relation* (we write $q \xrightarrow{a} q'$ instead of $(q, a, q') \in \delta$), q_I is the *initial state*, and $F \subseteq B$ is a set of accepting states. The automaton $\mathcal{B}$ is *deterministic* if for each $q \in B$ and each $a \in \Sigma$, there is at most one $q' \in B$ such that $q \xrightarrow{a} q'$.

The symbol Σ^ω denotes the set of all infinite words over the alphabet Σ. A *computation* of $\mathcal{B}$ on a word $w = w(0)w(1)\cdots \in \Sigma^\omega$ is a sequence $\omega = q_0, q_1, \ldots$ of states of $\mathcal{B}$ such that $q_0 = q_I$ and for all $i \geq 0$ we have $q_i \xrightarrow{w(i)} q_{i+1}$. A computation ω of $\mathcal{B}$ is *accepting* if a state of F occurs infinitely many times in ω. The automaton $\mathcal{B}$ accepts a word $w \in \Sigma^\omega$ if there exists an accepting computation of $\mathcal{B}$ on w. The set of all $w \in \Sigma^\omega$ accepted by $\mathcal{B}$ is denoted $\mathcal{L}(\mathcal{B})$.

The logic qPECTL* has the following syntax:

$$\Phi ::= a \mid \neg a \mid \mathcal{B}^{\sim\varrho}(\Phi_1, \cdots, \Phi_n)$$

Here a ranges over the set Ap of atomic propositions, $\sim\!\varrho \in \{=1, <1, >0, =0\}$, $n \geq 1$, $\mathcal{B}$ is a Büchi automaton over an alphabet $\Sigma \subseteq 2^{\{1,\ldots,n\}}$, and each Φ_i is a qPECTL* formula.

The semantics of qPECTL* formulae is defined below. Let $M = (S, \rightarrow, \mathit{Prob})$ be a Markov chain and let $\nu : Ap \rightarrow 2^S$ be a valuation. We define $s \models^\nu a$ iff $s \in \nu(a)$, and

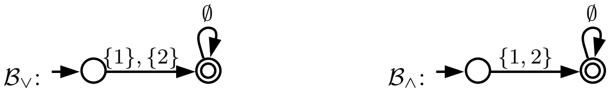
$s \models^{\nu} \neg a$ iff $s \notin \nu(a)$. The semantics of a qPECTL* formula $\Phi = \mathcal{B}^{\sim\varrho}(\Phi_1, \cdots, \Phi_n)$, where $\mathcal{B}$ is a Büchi automaton with the alphabet $\Sigma \subseteq 2^{\{1,\dots,n\}}$, is defined as follows: First, we can assume that the semantics of the qPECTL* formulae $\Phi_1, \dots, \Phi_n$ has already been defined. For every state s of M, let $Run(s, \Phi)$ be the set of all runs $w \in Run(s)$ satisfying the following condition: There is a word $v \in \mathcal{L}(\mathcal{B})$ such that for all $i \geq 0$ and all $k \in v(i)$ holds $w(i) \models^{\nu} \Phi_k$. We stipulate that $s \models^{\nu} \Phi$ if and only if $\mathcal{P}(Run(s, \Phi)) \sim \varrho$.

We say that formulae Φ and Ψ are *equivalent* ($\Phi \equiv \Psi$) iff for each state s of an arbitrary Markov chain M and for arbitrary valuation ν holds: $s \models^{\nu} \Phi$ iff $s \models^{\nu} \Psi$.

Remark 1. Note that once a formula $\mathcal{B}^{\sim\varrho}(\Phi_1, \dots, \Phi_n)$, a Markov chain M, and a valuation ν are fixed, we can say that $\mathcal{B}$ (or any automaton with the alphabet $\Sigma \subseteq 2^{\{1,\dots,n\}}$) accepts a run w of M if there is a word $v \in \mathcal{L}(\mathcal{B})$ such that for all $i \geq 0$ we have $\bigwedge_{k \in v(i)} w(i) \models^{\nu} \Phi_k$. Then, e.g., $\mathcal{P}(Run(s, \Phi))$ is the probability that $\mathcal{B}$ accepts a run of $Run(s)$. We can also say that the automaton $\mathcal{B}$ goes from a state q_0 to q_{i+1} after reading a finite path $s_0, \dots, s_i$ in M if there is a sequence $q_0, \dots, q_{i+1}$ of states of $\mathcal{B}$ and a word $X_0, \dots, X_i$ such that $q_j \overset{X_j}{\to} q_{j+1}$ and $\bigwedge_{k \in X_j} w(j) \models^{\nu} \Phi_k$ for all $0 \leq j \leq i$.

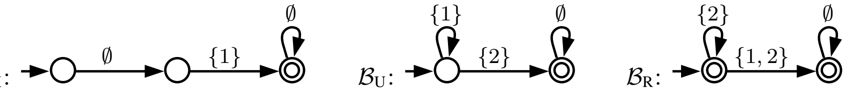
For computational purposes we assume that each formula is represented as a directed acyclic multigraph obtained from the parse tree of the formula by merging similar subtrees. For example, the formula $\mathcal{B}_1^{\sim\varrho}(\mathcal{B}_1^{\sim\varrho}(a, a, a), \mathcal{B}_1^{\sim\varrho}(a, a, a), \mathcal{B}_2^{\sim\varrho}(\mathcal{B}_1^{\sim\varrho}(a, a, a)))$ is represented by a multigraph with four nodes n_1, n_2, n_3, n_4 labeled with $\mathcal{B}_1^{\sim\varrho}, \mathcal{B}_2^{\sim\varrho}, \mathcal{B}_1^{\sim\varrho}$, a, respectively, and transitions: $n_1 \overset{1,2}{\to} n_3$ (here the numbers $1, 2$ stand for the first and the second argument), $n_1 \overset{3}{\to} n_2$, $n_2 \overset{1}{\to} n_3$, $n_3 \overset{1,2,3}{\to} n_4$. Here n_1 corresponds to the whole formula.

*Expressing other operators in qPECTL**. The logic qPECTL*, as defined above, is very powerful and succinct, and hence ideal for theoretical considerations. However, it is easier to express complex properties when we have some additional operators. We show that all operators of qPCTL can be expressed in qPECTL*. We define automata $\mathcal{B}_{\wedge}$, $\mathcal{B}_{\vee}$ as follows:



It is easy to see that formulae $\mathcal{B}_{\vee}^{=1}(\Phi_1, \Phi_2)$ and $\mathcal{B}_{\wedge}^{=1}(\Phi_1, \Phi_2)$ are equivalent to logical disjunction and conjunction, respectively, of Φ_1 and Φ_2. Hence, in what follows we write $\Phi_1 \vee \Phi_2$ and $\Phi_1 \wedge \Phi_2$ instead of $\mathcal{B}_{\vee}^{=1}(\Phi_1, \Phi_2)$ and $\mathcal{B}_{\wedge}^{=1}(\Phi_1, \Phi_2)$, respectively.

We also define Büchi automata representing 'next', 'until' and 'release' (the dual of 'until') operators:



We write $\mathrm{X}^{\sim\varrho}\Phi_1$, $\Phi_1\mathrm{U}^{\sim\varrho}\Phi_2$ and $\Phi_1\mathrm{R}^{\sim\varrho}\Phi_2$ instead of $\mathcal{B}_{\mathrm{X}}^{\sim\varrho}(\Phi_1)$, $\mathcal{B}_{\mathrm{U}}^{\sim\varrho}(\Phi_1, \Phi_2)$ and $\mathcal{B}_{\mathrm{R}}^{\sim\varrho}(\Phi_1, \Phi_2)$, respectively. We also define 'future' and 'globally' operators as follows:

Let $\mathtt{tt}$ and $\mathtt{ff}$ stand for $a \vee \neg a$ and $a \wedge \neg a$, respectively, for some $a \in Ap$. Let $\mathrm{F}^{\sim\varrho}\Phi$ stands for $\mathtt{tt}\mathrm{U}^{\sim\varrho}\Phi$, and let $\mathrm{G}^{\sim\varrho}\Phi$ stands for $\mathtt{ff}\mathrm{R}^{\sim\varrho}\Phi$.

Given a formula of the form $\mathcal{B}^{\sim\varrho}(\Phi_1,\ldots,\Phi_n)$, we write $\neg\mathcal{B}^{\sim\varrho}(\Phi_1,\ldots,\Phi_n)$ to stand for $\mathcal{B}^{\bowtie\varrho}(\Phi_1,\ldots,\Phi_n)$, where '$\bowtie\varrho$' is '$=1$', '$<1$', '$>0$', or '$=0$', depending on whether '$\bowtie\varrho$' is '<1', '$=1$', '$=0$', or '>0', respectively. This clearly corresponds to the logical operation of negation. Note that $\Phi_1\mathrm{R}^{\sim\varrho}\Phi_2$ is equivalent to $\neg(\neg\Phi_1\mathrm{U}^{\sim\varrho}\neg\Phi_2)$.

Now qPCTL is the fragment of qPECTL* consisting of all formulae of the following form:

$$\Phi ::= a \mid \neg a \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \wedge \Phi_2 \mid \mathrm{X}^{\sim\varrho}\Phi_1 \mid \Phi_1\mathrm{U}^{\sim\varrho}\Phi_2 \mid \Phi_1\mathrm{R}^{\sim\varrho}\Phi_2$$

Here $\sim\varrho$ ranges over $\{=1,=0,<1,>0\}$.

One can also show that all formulae of qPCTL* (for definition see, e.g., [5]) can be translated to equivalent qPECTL* formulae. This translation employs the algorithm for translating LTL formulae to Büchi automata (see, e.g., [15]) which results in a single exponential blow-up in the size of formulae.

*The logic detPECTL**. Now we define the *deterministic* fragment of qPECTL* (called detPECTL*), which generalizes the fragment $\mathcal{L}(\mathrm{F}^{=1},\mathrm{F}^{>0},\mathrm{G}^{=1})$ defined in [3] (see also Section 5). This fragment (together with Theorem 6) plays the crucial role in the proof of Theorem 8.

Given an automaton $\mathcal{B}$ with an alphabet $\Sigma \subseteq 2^{\{1,\ldots,n\}}$, we say that a state q of $\mathcal{B}$ is *terminal* iff there is a transition $q \xrightarrow{\emptyset} q$ and no transition of the form $q \xrightarrow{X} q'$ where $q \neq q'$ in $\mathcal{B}$. A formula Φ of qPECTL* is a detPECTL* formula if all subformulae of Φ of the form $\mathcal{B}^{\sim\varrho}(\Phi_1,\cdots,\Phi_n)$ satisfy the following conditions:

1. '$\sim\varrho$' is either '$=1$' or '>0';
2. if '$\sim\varrho$' is '>0', then all accepting states of $\mathcal{B}$ are terminal;
3. All states q of $\mathcal{B}$ satisfy the following condition: For *distinct nonterminal* states q' and q'' such that $q \xrightarrow{A} q'$ and $q \xrightarrow{B} q''$, we have that $\bigwedge_{i\in A\cup B}\Phi_i$ is *not* satisfied in any state of any Markov chain for any valuation (this must hold even for $A = B$).

Observe that $\mathrm{X}^{=1}$, $\mathrm{X}^{>0}$, $\mathrm{U}^{=1}$, $\mathrm{U}^{>0}$, and $\mathrm{R}^{=1}$ are operators of detPECTL*.

2.3 Games and Strategies

A $1\frac{1}{2}$-*player game* (or *Markov decision process*) is a tuple $G = (V, E, (V_\Box, V_\bigcirc), \mathit{Prob})$ where V is a finite set of *vertices*, $E \subseteq V \times V$ is a set of *transitions*, $(V_\Box, V_\bigcirc)$ is a partition of V, and Prob is a *probability assignment* which to each $v \in V_\bigcirc$ assigns a positive probability distribution on the set of its outgoing transitions. For technical convenience, we assume that each vertex has at least one outgoing transition.

The game is played by a player $\Box$ who selects the moves in the $V_\Box$ vertices, and a "virtual" player $\bigcirc$ who selects the moves in the $V_\bigcirc$ vertices according to the corresponding probability distribution.

A *strategy* for player $\Box$ is a function σ which to each $vs \in V^*V_\Box$ assigns a probability distribution on the set of outgoing transitions of s. We say that a strategy σ is *deterministic* if $\sigma(vs)$ is a Dirac distribution for each $vs \in V^*V_\Box$. Consistently with [1,11,3], we use HR and HD to denote the classes of all (history-dependent randomized) strategies and (history-dependent) deterministic strategies, respectively. A special type of strategies are strategies with *finite-memory*, which are formally defined as pairs $(\mathcal{A}, f)$ where $\mathcal{A} = (Q, V, \delta, q_0)$ is a deterministic finite-state automaton over the alphabet V of vertices and f is a function which to each pair $(q, s) \in Q \times V_\Box$ assigns a probability distribution on the set of outgoing transitions of s. The pair $(\mathcal{A}, f)$ determines a unique strategy $\sigma(\mathcal{A}, f)$ such that $\sigma(\mathcal{A}, f)(vs) = f(q, s)$, where $q = \delta(q_0, vs)$. Intuitively, the states of $\mathcal{A}$ represent a finite memory of size $|Q|$ where selected properties of the history of a play are stored. We denote FR and FD the classes of all finite-memory strategies and finite-memory deterministic strategies, respectively.

Each strategy σ for player $\Box$ determines a unique *play* of the game G, which is a Markov chain $G(\sigma)$ where V^+ is the set of states, and $vs \xrightarrow{x} vst$ iff $(s, t) \in E$ and one of the following conditions holds:

– $s \in V_\bigcirc$ and $\mathit{Prob}(s, t) = x$;
– $s \in V_\Box$ and $\sigma(vs)$ assigns x to (s, t).

For every $vs \in V^*V$ we denote $\mathit{last}(vs) = s$. For every run w of $G(\sigma)$ and every $i \geq 0$, we define $w[i] = \mathit{last}(w(i))$ (note that $w(i)$ is a finite sequence of vertices of the game G).

An *objective* is a pair (ν, Φ), where $\nu : Ap \to 2^V$ is a valuation and Φ a qPECTL* formula. Note that each valuation $\nu : Ap \to 2^V$ determines a valuation $\overline{\nu} : Ap \to 2^{V^+}$ defined by $\overline{\nu}(a) = \{vs \in V^*V \mid s \in \nu(a)\}$. For a given objective (ν, Φ), each state of $G(\sigma)$ either does or does not satisfy Φ. A (ν, Φ)*-winning strategy* for player $\Box$ in a vertex $s \in V$ is a strategy σ such that $s \models^\nu \Phi$. We are interested in the following problem:

Synthesis problem: Given a vertex s and an objective (ν, Φ),

is there a (ν, Φ)-winning strategy for player $\Box$ in s ?

Moreover, if the winning strategy exists, then return its finite representation.

Remark 2. Let σ be a finite-memory strategy determined by $(\mathcal{A}, f)$ where $\mathcal{A} = (Q, V, \delta, q_0)$. Observe, that the chain $G(\sigma)$ can be seen as an 'unfolding' of a finite Markov chain. Formally, let $\approx \subseteq V^+ \times V^+$ be an equivalence defined as follows: $u \approx v$ if and only if $\delta(q_0, u) = \delta(q_0, v)$ and $\mathit{last}(u) = \mathit{last}(v)$. Given $v \in V^+$ we denote $[v] = \{u \mid u \approx v\}$, the equivalence class of v, and we denote $V^+/\approx\, = \{[v] \mid v \in V^+\}$. Let us define a finite Markov chain $\bar{G}(\sigma)$ where $V^+/\approx$ is a set of states, and $[v] \xrightarrow{x} [vs]$ in $\bar{G}(\sigma)$ if and only if $v \xrightarrow{x} vs$ in $G(\sigma)$. Each valuation $\nu : Ap \to 2^V$ determines a valuation $\bar{\nu} : Ap \to 2^{V^+/\approx}$ defined by $\overline{\nu}(a) = \{[vs] \mid s \in \nu(a)\}$. Now, it is easy to verify that for every qPECTL* formula Φ, every valuation ν, and every state v of $G(\sigma)$, we have that $v \models^\nu \Phi$ iff $[v] \models^{\bar{\nu}} \Phi$.

3 The Synthesis Problem for detPECTL*

In this section we prove that the synthesis problem is decidable in exponential time for both HR and HD strategies and detPECTL* objectives. The proof follows similar lines as the analogous proof for HD strategies and the $\mathcal{L}(\mathrm{F}^{=1}, \mathrm{G}^{=1}, \mathrm{F}^{>0})$ fragment of qPCTL (see [3]). However, new technical difficulties arise from the use of automata connectives and randomization.

Similarly to [3], we reduce the synthesis problem for detPECTL* to the problem of solving $1\frac{1}{2}$-player games for a different type of objectives (and strategies) defined as follows. Let $G = (V, E, (V_\Box, V_\bigcirc), Prob)$ be a $1\frac{1}{2}$-player game. A *mixed* Büchi objective is a pair (P, O) where $P, O \subseteq V$. A strategy σ is (P, O)-winning in a vertex s iff *all* runs in $G(\sigma)$ initiated in s visit a vertex of P infinitely often, and *almost all* runs initiated in s visit a vertex of O infinitely often. To be able to control randomization in games we introduce a new restriction on strategies defined as follows: Given a set $\Re \subseteq V_\Box$, we say that a (HR) strategy σ is $\Re$-*must* iff for every $v \in V^*$, each $s \in \Re$ and each $(s, t) \in E$ we have $\sigma(vs)(s, t) > 0$, and for each $t \in V_\Box \setminus \Re$ we have that $\sigma(vt)$ is a Dirac distribution. Intuitively, a strategy is $\Re$-must if it always assigns non-zero probability to all successors of vertices of $\Re$ and behaves deterministically in vertices of $V_\Box \setminus \Re$.

Let us fix a game $G = (V, E, (V_\Box, V_\bigcirc), Prob)$, a vertex s_{in} of G, a detPECTL* formula Φ, and a valuation ν. The following lemma allows us to assume that the branching degree of all vertices of G is at most two (we sketch the proof in [4]).

Lemma 3. *There is a $1\frac{1}{2}$-player game $\bar{G}$, a vertex $\bar{s}_{in}$, a formula $\bar{\Phi}$, and a valuation $\bar{\nu}$ (computable in polynomial time), such that each vertex of $\bar{G}$ has at most two successors, and there is a (ν, Φ)-winning strategy in s_{in} iff there is a $(\bar{\nu}, \bar{\Phi})$-winning strategy in $\bar{s}_{in}$. Moreover, each (ν, Φ)-winning FR (FD) strategy in s_{in} can be polynomially translated to a $(\bar{\nu}, \bar{\Phi})$-winning FR (FD) strategy in $\bar{s}_{in}$, and vice versa.*

We construct a game G', a vertex s'_{in} of G', a mixed Büchi objective (P, O), and a set $\Re$, such that there is a (P, O)-winning $\Re$-must HR strategy in s'_{in} iff there is (ν, Φ)-winning HR strategy in s_{in}. The size of G' will be single exponential in the size of Φ and polynomial in the size of G.

To simplify our presentation we introduce some additional notation. We say that a Büchi automaton $\mathcal{B}$ *corresponds* to a formula Ψ if and only if Ψ is of the form $\mathcal{B}^{\sim\varrho}(\Phi_1, \ldots, \Phi_n)$ (for some $\sim\varrho$ and $\Phi_1, \ldots, \Phi_n$). For technical convenience, we assume that each Büchi automaton corresponds to at most one subformula of Φ and that all automata occurring in Φ have pairwise disjoint sets of states. Let $States$ denote the set of all states of all automata occurring in Φ, and let $States^{>0}$ and $States^{=1}$ denote sets of states of all automata that correspond to subformulae of Φ of the form $\mathcal{B}^{>0}(\Phi_1, \ldots, \Phi_n)$ and $\mathcal{B}^{=1}(\Phi_1, \ldots, \Phi_n)$, respectively. By $L(s)$ we denote the set of all literals (i.e., atomic propositions and negated atomic propositions) satisfied in the vertex s.

Now we present a formal definition of the game G'. An intuition behind the definition is given below.

Formal definition of G'. We define $G' = (V', E', (V'_\Box, V'_\bigcirc), Prob')$ where the set V' consists of vertices of the following three forms:

– f-vertices are of the form $(s, A)^f$, where $s \in V$ and $A \subseteq \mathit{States} \times \{\circ, \star\}$.
– g-vertices are of the form $(s, \mathcal{D})^g$, where

$$\mathcal{D} \subseteq \{(t, B) \mid (s, t) \in E, B \subseteq \mathit{States} \times \{\circ, \star\}\}$$

is a non-empty set, for each t there is at most one pair of the form (t, B) in $\mathcal{D}$, and if $s \in V_\bigcirc$ and $(s, t) \in E$, then $\mathcal{D}$ contains a pair of the form (t, B).
– distinguished vertices s'_{in} and dead.

To $V'_\bigcirc$ we put all g-vertices whose first component belongs to $V_\bigcirc$, and we put $V'_\Box = V' \setminus V'_\bigcirc$. To formally define E' we need some additional notation. Given a formula of the form $\Psi = \mathcal{B}^{\sim\varrho}(\Phi_1, \ldots, \Phi_n)$, we denote $\mathit{Rep}(\Psi)$ the tuple $(q_0, \star)$ where q_0 is the initial state of $\mathcal{B}$. Given a literal Ψ, we define $\mathit{Rep}(\Psi) = \Psi$. Given a transition $q \xrightarrow{X} q'$ of an automaton corresponding to a formula of the form $\mathcal{B}^{\sim\varrho}(\Phi_1, \ldots, \Phi_n)$, we denote $\mathit{Start}(q \xrightarrow{X} q')$ the set of all $\mathit{Rep}(\Phi_i)$ where $i \in X$. The set of transitions E' is defined as follows:

– $((s, A)^f, (s, \mathcal{D})^g) \in E'$ for *all* $(s, A)^f$ and $(s, \mathcal{D})^g$ satisfying the following: for *every* $(q, x) \in A$ there *exists* (q', x') satisfying $q \xrightarrow{X} q'$ and $\mathit{Start}(q \xrightarrow{X} q') \subseteq A \cup L(s)$ and

$$x' = \begin{cases} \star & \text{if } q' \text{ is accepting;} \\ \circ & \text{if } q' \in \mathit{States}^{=1} \text{ is not accepting and } A \cap (\mathit{States}^{=1} \times \{\circ\}) = \emptyset; \\ \circ & \text{if } q' \in \mathit{States}^{>0} \text{ is not accepting and } A \cap (\mathit{States}^{>0} \times \{\circ\}) = \emptyset; \\ x & \text{otherwise.} \end{cases}$$

and either $(q', x') \in \bigcap_{(t,B) \in \mathcal{D}} B$ or $(q', x') \in \bigcup_{(t,B) \in \mathcal{D}} B$, depending on whether $q \in \mathit{States}^{=1}$ or $q \in \mathit{States}^{>0}$, respectively.
– $((s, A)^f, \mathit{dead}) \in E'$ for all f-vertices, and $(\mathit{dead}, \mathit{dead}) \in E'$;
– $((s, \mathcal{D})^g, (t, B)^f) \in E'$ for all $(t, B) \in \mathcal{D}$;
– $(s'_{in}, (s_{in}, A)^f) \in E'$ for all $A \subseteq \mathit{States} \times \{\circ, \star\}$ satisfying $\mathit{Rep}(\Phi) \in A$ (here we assume that Φ is not a literal, because otherwise the synthesis problem is trivially solved)

We define $\mathit{Prob}'((s, \mathcal{D})^g, (t, B)^f) = \mathit{Prob}(s, t)$ whenever $s \in V_\bigcirc$ and $(t, B) \in \mathcal{D}$.

Let P be the set of all vertices of the form $(s, A)^f$ such that A does not contain any pair of the form $(q, \circ)$ where $q \in \mathit{States}^{>0}$. Let O be the set of all vertices of the form $(s, A)^f$ such that A does not contain any pair of the form $(q, \circ)$ where $q \in \mathit{States}^{=1}$.

Let $\Re$ be the set of all g-vertices of the form $(s, \mathcal{D})^g$ where $s \in V_\Box$.

Intuition behind G'. The game G' simulates the game G (in the first component of vertices) and at the same time maintains some information about subformulae of Φ (the second component of vertices). Each step of the simulated play of G corresponds to two steps in G'. The first step, going from the current f-vertex to a g-vertex, updates the information about Φ. The next one, going from the g-vertex to an f-vertex, simulates a move in G.

While playing the game G', player $\Box$ simulates a play of the game G and at the same time simulates computations of Büchi automata corresponding to subformulae of Φ, verifying that these subformulae are satisfied in appropriate places in the simulated play. Given an f-vertex $(s, A)^f$, every pair $(q, x) \in A$ represents a running instance of an automaton which is in the state q. (Here x maintains the information whether this particular instance recently entered an accepting state.)

Going from the f-vertex $(s, A)^f$ to a g-vertex $(s, \{(t_1, B_1), (t_2, B_2)\})^g$, player $\Box$ simulates one computational step for each running instance $(q, x) \in A$. More concretely, let $(q, x) \in A$ where q is a state of an automaton corresponding to $\mathcal{B}^{\sim\varrho}(\Phi_1, \ldots, \Phi_n)$. Player $\Box$ chooses a transition $q \xrightarrow{X} q'$ such that $Start(q \xrightarrow{X} q') \subseteq A \cup L(s)$, which intuitively means that the running instances of automata corresponding to formulae of $\{\Phi_i \mid i \in X\}$ have already been initiated in $(s, A)^f$. Then (q', x') (here x' is an appropriate update of x) is put either to both B_1, B_2 or to at least one of them, depending on whether '$\sim\varrho$' is either '$=1$' or '>0', respectively. Note that in the case '$=1$', the simulated computation goes to the same state q' for both successors. To ensure correctness of the simulation, we need Φ to be a detPECTL* formula (intuitively this means that there is at most one 'correct' non-terminal successor q' of q after reading the current state).

Note that the definition of x' and the winning objective (P, O) ensure that *all* running instances of automata corresponding to formulae of the form $\mathcal{B}^{=1}(\ldots)$ are almost surely accepting (i.e., enter an accepting state infinitely many times), and that *all* running instances of automata corresponding to formulae of the form $\mathcal{B}^{>0}(\ldots)$ are surely accepting (i.e., enter a terminal accepting state). Note that the sets B_1 and B_2 may, in addition to the obligatory contents, contain arbitrary pairs from $States \times \{\circ, \star\}$. This may be used by player $\Box$ to initiate new running instances needed in the next simulation step to perform transitions of Büchi automata (see above).

Finally, going from the g-vertex $(s, \{(t_1, B_1), (t_2, B_2)\})^g$ to an f-vertex, player $\Box$ randomly chooses one of the successors $(t_1, B_1)^f, (t_2, B_2)^f$, by which he chooses a successor in the simulated play. The $\Re$-must restriction ensures that each of the successors is chosen with non-zero probability, which prevents player $\Box$ from erasing pairs of $States^{>0} \times \{\circ, \star\}$.

The following lemma is proved in [4].

Lemma 4. *There is a (ν, Φ)-winning HR strategy in s_{in} if and only if there is a (P, O)-winning $\Re$-must HR strategy in s'_{in}. Moreover, each (P, O)-winning $\Re$-must FR strategy $(\mathcal{A}, f)$ in s'_{in} induces a (ν, Φ)-winning FR strategy in s_{in} computable in time polynomial in the size of $(\mathcal{A}, f)$.*

It has been shown in [3] that the existence of a winning HD strategy in $1\frac{1}{2}$-player games with mixed Büchi objectives is decidable in polynomial time, and moreover, that the existence of a winning HD strategy in such games implies the existence of a winning FD strategy computable in polynomial time. By a slight modification of the proof from [3] we obtain the following analogy for $\Re$-must HR strategies:

Lemma 5. *The existence of a winning $\Re$-must HR strategy in $1\frac{1}{2}$-player games with mixed Büchi objectives is decidable in polynomial time. Moreover, in these games, the* *existence of a winning $\Re$-must HR strategy implies the existence of a winning $\Re$-must FR strategy computable in polynomial time.*

Applying Lemma 4 and Lemma 5 we obtain the following theorem.

Theorem 6. *The existence of a winning HR (HD) strategy in $1\frac{1}{2}$-player games with detPECTL* objectives is decidable in time exponential in the size of formulae and polynomial in the size of games. Moreover, in these games, the existence of a winning HR (HD) strategy implies the existence of a winning FR (FD) strategy computable in time exponential in the size of formulae and polynomial in the size of games.*

Proof (Sketch). For HR strategies, the result follows immediately from Lemma 4 and Lemma 5. For HD strategies, it suffices to slightly modify the construction of the game G' by erasing all g-vertices $(s, \mathcal{D})^g$ such that $s \in V_\Box$ and $|\mathcal{D}| > 1$. Now for $\Re = \emptyset$, each $\Re$-must strategy in s'_{in} is deterministic. An inspection of the proof of Lemma 4 reveals that the lemma remains valid even for deterministic strategies. Now using Lemma 5 we obtain the desired result.

4 The Synthesis Problem for qPECTL* and Finite-Memory Strategies

In this section we show how to solve the synthesis problem for qPECTL* and finite-memory strategies. We show that, in fact, the logic qPECTL* and its fragment detPECTL* are expressively equivalent over finite Markov chains, and then obtain the solution to the synthesis problem as an immediate corollary of our previous results. To formally capture this equivalence, we write $\Phi \equiv_{fin} \Psi$ whenever for arbitrary state s of arbitrary *finite* Markov chain and arbitrary valuation ν, holds $s \models^\nu \Phi$ iff $s \models^\nu \Psi$. The main aim of this section is formalized by the following theorem.

Theorem 7. *For every qPECTL* formula Φ there is a detPECTL* formula Ψ, computable in exponential time, such that $\Phi \equiv_{fin} \Psi$. Moreover, if Φ is a qPCTL formula, then Ψ is computable in polynomial time.*

An immediate corollary of Theorem 7, Theorem 6, and Remark 2 is the following

Theorem 8. *For both FR and FD strategies, the synthesis problem for qPCTL, qPECTL*, and qPCTL* objectives can be solved in single exponential, double exponential, and triple exponential time, respectively. Moreover, in all these cases, the synthesis problem can be solved in time polynomial in the size of games.*

The rest of this section is devoted to the proof of Theorem 7. Let us fix a qPECTL* formula Φ. First, observe that we may assume (w.l.o.g.) that for each subformula of Φ of the form $\mathcal{B}^{\sim\varrho}(\Phi_1, \ldots, \Phi_n)$, every state s of an arbitrary Markov chain and an arbitrary valuation ν, there is *exactly one* letter A in the alphabet of $\mathcal{B}$ such that $s \models^\nu \bigwedge_{i \in A} \Phi_i$. Indeed, if Φ does not have this property, it suffices to substitute each subformula of Φ of the form $\mathcal{B}^{\sim\varrho}(\Phi_1, \ldots, \Phi_n)$ with a formula of the form $\bar{\mathcal{B}}^{\sim\varrho}(\Phi_1, \ldots, \Phi_n, \neg\Phi_1, \ldots, \neg\Phi_n)$, where $\bar{\mathcal{B}}$ is obtained from $\mathcal{B}$ by substituting each transition of the form $q \xrightarrow{A} q'$ with all transitions of the form $q \overset{A' \cup A''}{\rightarrow} q'$ where $A \subseteq A' \subseteq \{1, \dots, n\}$ and $A'' = \{m + n | m \in \{1, \dots, n\} \setminus A'\}$, and consequently making the transition function total. Observe that although the alphabet of $\bar{\mathcal{B}}$ may be exponentially larger than the alphabet of $\mathcal{B}$, the number of states increases only by 1 due to making the transition function total. Observe also that the size of the (graph representing) resulting formula is polynomial in the size of Φ.

Now, to determinize the formula Φ, it suffices to determinize (syntactically) transition relations of all Büchi automata in Φ. However, the problem is that deterministic Büchi automata are strictly weaker than non-deterministic Büchi automata. We solve this problem by first translating the Büchi automata to equivalent deterministic Rabin automata (using results of [14]) and then encoding the deterministic Rabin automata to detPECTL* formulae.

First, let us formally define the notion of deterministic Rabin automata. A *deterministic Rabin automaton* $\mathcal{R}$ is a tuple $(Q, \Sigma, \gamma, q_0, Acc)$, where Q is a finite set of states, Σ is an input alphabet, $\gamma : Q \times \Sigma \to Q$ is a transition function, q_0 is an initial state, and $Acc = \{(C_1, D_1), \dots, (C_k, D_k)\}$, where $C_1, \dots, C_k, D_1, \dots, D_k \subseteq Q$, specifies the acceptance condition. A word $w \in \Sigma^\omega$ is accepted by $\mathcal{R}$ iff there is a sequence $\omega = q_0, q_1, \dots$ of states of $\mathcal{R}$ such that for all $i \geq 0$ we have $\gamma(q_i, w(i)) = q_{i+1}$, and there is $j \in \{1, \dots, k\}$ such that some state of C_j occurs infinitely often in ω and no state of D_j occurs infinitely often in ω. We denote $\mathcal{L}(\mathcal{R})$ the set of all words accepted by $\mathcal{R}$. The following proposition was proved in [14].

Theorem 9 ([14]). *Given a Büchi automaton $\mathcal{B} = (B, \Sigma, \delta, q_I, F)$ there is an effectively computable deterministic Rabin automaton $\mathcal{R} = (Q, \Sigma, \gamma, q_0, Acc)$ such that $\mathcal{L}(\mathcal{R}) = \mathcal{L}(\mathcal{B})$, $|R| = 2^{\mathcal{O}(|B| \log |B|)}$ and $|Acc| = \mathcal{O}(|B|)$.*

Now let $\mathcal{B}^{\sim\varrho}(\Phi_1, \dots, \Phi_n)$ be a subformula of Φ and let us assume that $\Phi_1, \dots, \Phi_n$ are already detPECTL* formulae. Let us denote Σ the alphabet of $\mathcal{B}$, and let us fix a Rabin automaton $\mathcal{R} = (Q, \Sigma, \gamma, q_0, Acc)$, where $Acc = \{(C_1, D_1), \dots, (C_k, D_k)\}$, such that $\mathcal{L}(\mathcal{R}) = \mathcal{L}(\mathcal{B})$. Let us assume (w.l.o.g.) that the transition function of $\mathcal{R}$ is total.

Let us first assume that '$\sim\varrho$' is either of the form '>0' or '=1'. Based on $\mathcal{R}$, we define deterministic Büchi automata $\mathcal{B}_{fin}$ and $\mathcal{B}_{q,i}$, for all $q \in Q$ and $1 \leq i \leq k$, such that

$$\mathcal{B}^{\sim\varrho}(\Phi_1, \dots, \Phi_n) \equiv_{fin} \mathcal{B}_{fin}^{\sim\varrho}(\Phi_1, \dots, \Phi_n, \Psi_1, \dots, \Psi_\ell)$$

where each Ψ_j is of the form $\mathcal{B}_{q,i}^{=1}(\Phi_1, \dots, \Phi_n)$ for one of the automata $\mathcal{B}_{q,i}$ (i.e., $\ell = |Q| \cdot k$). In what follows we denote $index(q, i)$ the number j such that Ψ_j is the formula $\mathcal{B}_{q,i}^{=1}(\Phi_1, \dots, \Phi_n)$.

Before we formally define $\mathcal{B}_{fin}$ and $\mathcal{B}_{q,i}$, let us explain the intuition behind the definition. Let us fix a state s_0 of a finite Markov chain M and a valuation ν, and let us assume that $\mathcal{B}$ (and hence also $\mathcal{R}$) accepts a run of $Run(s_0)$ (see Remark 1) with a probability greater than 0 (the explanation is analogous for the probability =1). Because M is finite, there is a finite path $v \in FPath(s_0)$ such that $\mathcal{R}$ accepts almost all runs of $Run(v)$. Here, however, using basic results of the theory of finite Markov chains, one can say even more. There is a finite path $v \in FPath(s_0)$ such that almost all

$w \in Run(v)$ satisfy the following condition: the automaton $\mathcal{R}$, after reading the prefix v of w, enters a state of C_j infinitely often and no state of D_j at all, for a suitable j. We define $\mathcal{B}_{fin}$ and $\mathcal{B}_{q,i}$ so that $\mathcal{B}_{fin}^{>0}(\Phi_1, \ldots, \Phi_n, \Psi_1, \ldots, \Psi_\ell)$ expresses precisely this property.

The automata $\mathcal{B}_{fin}$ and $\mathcal{B}_{q,i}$ are formally defined as follows. Let $\mathcal{B}_{fin} = (Q \cup \{q_a\}, \Sigma \cup T, \delta_{fin}, q_0, \{q_a\})$, where $T = \{\{n+1\}, \ldots, \{n+\ell\}\}$, and transitions of $\mathcal{B}$ are defined as follows:

- $q \xrightarrow{A} q'$ for all $A \in \Sigma$ and $q, q' \in Q$ such that $\gamma(q, A) = q'$;
- $q \xrightarrow{\{n+index(q,i)\}} q_a$ for all $q \in Q$ and all $1 \le i \le k$;
- $q_a \xrightarrow{\emptyset} q_a$;
- nothing else is a transition.

We define $\mathcal{B}_{q,i} = (Q, \Sigma, \delta_{q,i}, q, C_i)$ where transitions are defined as follows: for all $q \in Q$ and all $A \in \Sigma$, we define $q \xrightarrow{A} q'$ if and only if $q, q' \notin D_i$ and $\gamma(q, A) = q'$ (i.e., there are no transitions leaving or entering states of D_i).

Lemma 10. *If '$\sim\varrho$' is either of the form '>0' or '$=1$', then*

$$\mathcal{B}^{\sim\varrho}(\Phi_1, \ldots, \Phi_n) \equiv_{fin} \mathcal{B}_{fin}^{\sim\varrho}(\Phi_1, \ldots, \Phi_n, \Psi_1, \ldots, \Psi_\ell)$$

Moreover, the right hand side formula is in detPECTL*.

Proof (Sketch). The fact that $\mathcal{B}_{fin}^{\sim\varrho}(\Phi_1, \ldots, \Phi_n, \Psi_1, \ldots, \Psi_\ell)$ is a detPECTL* formula follows immediately from our assumption about Φ and from the fact that the automata $\mathcal{B}_{fin}$ and $\mathcal{B}_{q,i}$ are obtained from the deterministic automaton $\mathcal{R}$ either by deleting transitions or by adding transitions to the newly added terminal state q_a.

It remains to prove the equivalence. Let us fix a finite Markov chain M with the set of states S, a state s_0 of M, and a valuation ν. Observe that for every state s of M there is exactly one $A_s \in \Sigma$ such that $s \models \bigwedge_{i \in A_s} \Phi_i$. Now an automaton with the alphabet Σ accepts a run $s_0, s_1, \ldots$ of M if it accepts the word $A_{s_0} A_{s_1} \cdots$.

First, let us assume that $s_0 \models \mathcal{B}_{fin}^{\sim\varrho}(\Phi_1, \ldots, \Phi_n, \Psi_1, \ldots, \Psi_\ell)$. It follows immediately from the definitions that, with probability $\sim\varrho$, there is a path $s_0, s_1, \ldots, s_i, s_{i+1}$ in M satisfying the following conditions:

- the automaton $\mathcal{B}_{fin}$ (and hence also the automaton $\mathcal{R}$) moves from its initial state to a state r after reading the word $A_{s_0} \cdots A_{s_i}$;
- $s_{i+1} \models \mathcal{B}_{r,j}^{=1}(\Phi_1, \ldots, \Phi_n)$ for some $1 \le j \le k$ (and hence almost all runs of $Run(s_{i+1})$ are accepted by the Rabin automaton $\mathcal{R}$ initiated in r).

However, this immediately implies that, with probability $\sim\varrho$, the automaton $\mathcal{R}$ (and hence the automaton $\mathcal{B}$) accepts a run of $Run(s_0)$.

For the opposite direction, let $M \times \mathcal{R}$ be a Markov chain (the synchronous product of M and $\mathcal{R}$) whose set of states is $S \times Q$ and transitions are defined as follows: $(s, q) \xrightarrow{x} (t, r)$ iff $s \xrightarrow{x} t$ and $q \xrightarrow{A_s} r$. Given $1 \le j \le k$, we say that a BSCC C of M is j-accepting iff a state of C_j occurs in (the second component of a state of) C and no state of D_j occurs in C. Basic results of the theory of Markov chains imply that the probability measure of all runs of $Run(s)$ accepted by $\mathcal{R}$ (hence also by $\mathcal{B}$) is equal to the probability of reaching some j-accepting BSCC of $M \times \mathcal{R}$. Thus, if $s_0 \models \mathcal{B}^{\sim\varrho}(\Phi_1, \ldots, \Phi_n)$, then, with probability $\sim\varrho$, there is a path $(s_0, q_0), \ldots, (s_i, q_i)$ in $M \times \mathcal{R}$ such that (s_i, q_i) belongs to a j-accepting BSCC. It is easy to show that $s_i \models \mathcal{B}^{=1}_{q_i,j}(\Phi_1, \ldots, \Phi_n)$ and $\mathcal{B}_{fin}$ moves from q_0 to q_a after reading the word $A_{s_0}, \ldots, A_{s_{i-1}}$, $\{n + index(q_i, j)\}$. This implies that $s_0 \models \mathcal{B}^{\sim\varrho}_{fin}(\Phi_1, \ldots, \Phi_n, \Psi_1, \ldots, \Psi_\ell)$. $\square$

Now, let us consider the case where '$\sim\varrho$' is either of the form '=0' or '<1'. We denote $\widehat{\sim\varrho}$ the 'dual' of '$\sim\varrho$', i.e., $\widehat{=0}$ is '=1', and $\widehat{<1}$ is '>0'. Using very similar arguments, we prove the following analogy of Lemma 10 (a proof can be found in [4]).

Lemma 11. *If '$\sim\varrho$' is either of the form '=0' or '<1', then there are Büchi automata $\mathcal{B}_{fin}$ and $\mathcal{B}_{q,i}$, for all $q \in Q$ and $1 \leq i \leq k$, such that*

$$\mathcal{B}^{\sim\varrho}(\Phi_1, \ldots, \Phi_n) \equiv_{fin} \mathcal{B}^{\widehat{\sim\varrho}}_{fin}(\Phi_1, \ldots, \Phi_n, \Psi_1, \ldots, \Psi_\ell)$$

where each Ψ_j is of the form $\mathcal{B}^{=1}_{q,i}(\Phi_1, \ldots, \Phi_n)$ for one of the automata $\mathcal{B}_{q,i}$ (i.e., $\ell = |Q| \cdot k$). Moreover, the right hand side formula is in detPECTL.*

Using Lemma 10 and Lemma 11 one can easily design an algorithm which transforms the formula Φ to a detPECTL* formula Ψ using appropriate substitutions in a 'bottom-up' manner.

For general qPECTL* formulae, the time complexity of the algorithm is single exponential in the size of Φ. Indeed, by [14] the Rabin automaton $\mathcal{R}$ can be computed in time exponential in the number of states of $\mathcal{B}$ and polynomial in the size of the alphabet, and consequently the automata $\mathcal{B}_{fin}$ and $\mathcal{B}_{q,i}$ are computable in single exponential time. It follows that the formula Ψ is computable in exponential time (here we make use of the representation of Ψ as a directed acyclic graph, i.e., we assume that several occurrences of the same subformula are represented by one vertex).

Now observe that in the case of PCTL formulae, there are only five distinct Büchi automata used to define Boolean connectives and operators X, U, and R. It follows that the corresponding Rabin automata have bounded size, and thus each PCTL formula can be translated to a detPECTL* formula in polynomial time. This finishes the proof of Theorem 7.

5 qPCTL and Finite-Memory Strategies

In this section we study the power of finite-memory strategies w.r.t. the synthesis problem for $1\frac{1}{2}$-player games and qPCTL objectives.

Given $\mathcal{Y} \subseteq \{\mathrm{X}^{\sim\varrho}, \mathrm{F}^{\sim\varrho}, \mathrm{G}^{\sim\varrho} \mid \sim\varrho \in \{=0, >0, <1, =1\}\}$, we denote $\mathcal{L}(\mathcal{Y})$ the fragment of qPCTL which consists of formulae of the following form:

$$\Phi ::= a \mid \neg a \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \wedge \Phi_2 \mid \mathrm{Y}^{\sim\varrho}\, \Phi_1$$

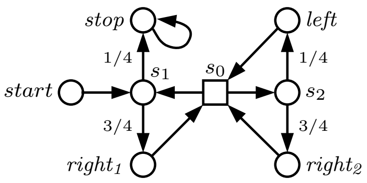
where $\mathrm{Y}^{\sim\varrho} \in \mathcal{Y}$. For example, the fragment $\mathcal{L}(\{\mathrm{F}^{=1}, \mathrm{G}^{=1}, \mathrm{F}^{>0}\})$ (we usually omit the set brackets and write $\mathcal{L}(\mathrm{F}^{=1}, \mathrm{G}^{=1}, \mathrm{F}^{>0})$) is the fragment of qPCTL whose formulae are built up from literals using conjunction, disjunction, and three temporal operators

$\mathrm{F}^{=1}, \mathrm{G}^{=1}, \mathrm{F}^{>0}$ (there is no operation of complement). Note that we work with the operators $\mathrm{F}^{\sim\varrho}$ and $\mathrm{G}^{\sim\varrho}$ only for simplicity: *All* results of this section remain valid even if one replaces $\mathrm{F}^{\sim\varrho}$ with $\mathrm{U}^{\sim\varrho}$, and $\mathrm{G}^{\sim\varrho}$ with $\mathrm{R}^{\sim\varrho}$.

Definition 12. *A fragment $\mathcal{L}(\mathcal{Y})$ is* finitely determined *if for every formula Φ of $\mathcal{L}(\mathcal{Y})$, arbitrary vertex s of a $1\frac{1}{2}$-player game, and arbitrary valuation ν, the following holds: If there is a (ν,Φ)-winning strategy in s, then there is a (ν,Φ)-winning* finite-memory *strategy in s.*

First, we show which fragments are *not* finitely determined. Then we prove that no finitely determined fragment is more expressive than the fragment $\mathcal{L}(\mathrm{X}^{=1}, \mathrm{X}^{>0}, \mathrm{G}^{=1}, \mathrm{F}^{=1}, \mathrm{F}^{>0})$.

It has been shown in [3] that $\mathcal{L}(\mathrm{G}^{>0}, \mathrm{F}^{>0})$ is not finitely determined. We extend this result and show that also $\mathcal{L}(\mathrm{G}^{>0})$ is not finitely determined. Let us consider a game G depicted in the following figure (it is very similar to the corresponding game from [3]):



We use names of vertices as atomic propositions with an obvious semantics. Let us denote $\Phi = \mathrm{G}^{>0}(\neg stop \wedge (\neg left \vee \mathrm{G}^{>0}\neg right_2))$. The proof of the following lemma is presented in [4].

Lemma 13. *There is a (ν,Φ)-winning HD strategy in $start$. There is* no *(ν,Φ)-winning FR strategy in $start$.*

We continue by proving that also $\mathcal{L}(\mathrm{G}^{=0})$ is not finitely determined. Let us consider a formula $\Psi = \mathrm{G}^{>0}(\neg stop \wedge (\neg left \vee \mathrm{F}^{=1}\mathrm{G}^{>0}\neg right_2))$ and the game G defined above. It is easy to show, using arguments similar to the proof of Lemma 13, that there is a (ν,Ψ)-winning HD strategy in $start$, and no (ν,Ψ)-winning FR strategy in $start$. Now we transform Ψ to a $\mathcal{L}(\mathrm{G}^{=0})$ formula Ψ' such that for an arbitrary strategy we have $start \models^{\nu} \Psi$ iff $start \models^{\nu} \Psi'$. Using obvious equivalences $\neg\mathrm{G}^{>0}\phi \equiv \mathrm{G}^{=0}\phi$ and $\mathrm{F}^{=1}\phi \equiv \mathrm{G}^{=0}\neg\phi$, one can easily show that Ψ is equivalent to $\neg\chi$ where $\chi = \mathrm{G}^{=0}(\neg stop \wedge (\neg left \vee \mathrm{G}^{=0}\mathrm{G}^{=0}\neg right_2))$. Now it is easy to verify that $start \models^{\nu} \neg\chi$ if and only if $start \models^{\nu} \mathrm{G}^{=0}(\neg start \vee \chi)$. It follows that $\mathcal{L}(\mathrm{G}^{=0})$ is not finitely determined.

Next, let us consider the fragment $\mathcal{L}(\mathrm{F}^{<1})$. Using the equivalence $\mathrm{G}^{>0}\phi \equiv \mathrm{F}^{<1}\neg\phi$, one can easily show that Ψ is equivalent to $\mathrm{F}^{<1}(stop \vee (left \wedge \mathrm{F}^{<1}\mathrm{F}^{<1}right_2))$. It follows that $\mathcal{L}(\mathrm{F}^{<1})$ is not finitely determined.

The last fragments we analyze are $\mathcal{L}(\mathrm{X}^{=0}, \mathrm{F}^{=1})$, $\mathcal{L}(\mathrm{X}^{<1}, \mathrm{F}^{=1})$, $\mathcal{L}(\mathrm{F}^{=0}, \mathrm{F}^{=1})$ and $\mathcal{L}(\mathrm{G}^{<1}, \mathrm{F}^{=1})$. Using the equivalences $\mathrm{F}^{=1}\phi \equiv \mathrm{G}^{=0}\neg\phi$ and $\neg\mathrm{F}^{=1}\phi \equiv \mathrm{G}^{>0}\neg\phi$ one can show that $\Phi = \mathrm{G}^{>0}(\neg stop \wedge (\neg left \vee \mathrm{G}^{>0}\neg right_2))$ is equivalent to $\neg\chi$ where $\chi = \mathrm{F}^{=1}(stop \vee (left \wedge \mathrm{F}^{=1}right_2))$. Now, using a similar trick as above, we obtain $start \models^{\nu} \neg\chi$ iff $start \models^{\nu} \mathrm{X}^{=0}(\chi)$ iff $start \models^{\nu} \mathrm{X}^{<1}(\chi)$ iff $start \models^{\nu} \mathrm{F}^{=0}(start \wedge \chi)$ iff $start \models^{\nu} \mathrm{G}^{<1}(\neg start \vee \chi)$.

Now we can give a complete classification of finitely determined fragments.

Lemma 14. *The fragment* $\mathcal{L}_1 = \mathcal{L}(\mathrm{X}^{=1}, \mathrm{X}^{<1}, \mathrm{X}^{>0}, \mathrm{X}^{=0}, \mathrm{G}^{=1}, \mathrm{F}^{>0}, \mathrm{F}^{=0}, \mathrm{G}^{<1})$ *and the fragment* $\mathcal{L}_2 = \mathcal{L}(\mathrm{X}^{=1}, \mathrm{X}^{>0}, \mathrm{G}^{=1}, \mathrm{F}^{>0}, \mathrm{F}^{=1})$ *are maximal (w.r.t. inclusion) finitely determined fragments.*

Proof. We have proved above that the fragments $\mathcal{L}(\mathrm{G}^{>0})$, $\mathcal{L}(\mathrm{G}^{=0})$, $\mathcal{L}(\mathrm{F}^{<1})$, $\mathcal{L}(\mathrm{X}^{=0}, \mathrm{F}^{=1})$, $\mathcal{L}(\mathrm{X}^{<1}, \mathrm{F}^{=1})$, $\mathcal{L}(\mathrm{F}^{=0}, \mathrm{F}^{=1})$ and $\mathcal{L}(\mathrm{G}^{<1}, \mathrm{F}^{=1})$ are not finitely determined. Clearly, any fragment which contains one of these fragments is not finitely determined. On the other hand, it follows from Theorem 6 that the fragment $\mathcal{L}_2$ *is* finitely determined. By a close inspection of various possibilities, we obtain that the only fragment we have not yet classified is $\mathcal{L}_1$ (and some of its subsets).

However, we show that each formula of $\mathcal{L}_1$ can efficiently be translated to $\mathcal{L}_2$, which implies that $\mathcal{L}_1$ is finitely determined. Let Ψ be a formula of $\mathcal{L}_1$. First, using the equivalences $\mathrm{X}^{=0}\Phi \equiv \mathrm{X}^{=1}\neg\Phi$, $\mathrm{X}^{<1}\Phi \equiv \mathrm{X}^{>0}\neg\Phi$, $\mathrm{F}^{=0}\Phi \equiv \mathrm{G}^{=1}\neg\Phi$, $\mathrm{G}^{<1}\Phi \equiv \mathrm{F}^{>0}\neg\Phi$, one can remove operators $\mathrm{X}^{=0}$, $\mathrm{X}^{<1}$, $\mathrm{F}^{=0}$, $\mathrm{G}^{<1}$ (introducing, however, some negations to the formula). The negations introduced in the previous step can be pushed to atomic propositions using equivalences $\neg\mathrm{X}^{=1}\phi \equiv \mathrm{X}^{>0}\neg\phi$, $\neg\mathrm{X}^{>0}\phi \equiv \mathrm{X}^{=1}\neg\phi$, $\neg\mathrm{G}^{=1}\phi \equiv \mathrm{F}^{>0}\neg\phi$, $\neg\mathrm{G}^{>0}\phi \equiv \mathrm{F}^{=1}\neg\phi$. □

Corollary 15. *A fragment* $\mathcal{L}(\mathcal{Y})$*, where* $\mathcal{Y} \subseteq \{\mathrm{X}^{\sim\varrho}, \mathrm{F}^{\sim\varrho}, \mathrm{G}^{\sim\varrho} \mid {\sim}\varrho \in \{=0, >0, <1, =1\}\}$*, is finitely determined if and only if each formula of* $\mathcal{L}(\mathcal{Y})$ *can be efficiently translated to an equivalent formula of* $\mathcal{L}(\mathrm{X}^{=1}, \mathrm{X}^{>0}, \mathrm{G}^{=1}, \mathrm{F}^{=1}, \mathrm{F}^{>0})$.

References

1. Baier, C., Größer, M., Leucker, M., Bollig, B., Ciesinski, F.: Controller synthesis for probabilistic systems. In: Proceedings of IFIP TCS'2004, Kluwer, Dordrecht (2004)
2. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed.) Foundations of Software Technology and Theoretical Computer Science. LNCS, vol. 1026, pp. 499–513. Springer, Heidelberg (1995)
3. Brázdil, T., Brožek, V., Forejt, V., Kučera, A.: Stochastic games with branching-time winning objectives. In: Proceedings of LICS 2006, IEEE Computer Society Press, Los Alamitos (2006)
4. Brázdil, T., Forejt, V.: Strategy synthesis for Markov decision processes and branching-time logics. Technical report FIMU-RS-2007-03 (2007)
5. Brázdil, T., Kučera, A., Stražovský, O.: On the decidability of temporal properties of probabilistic pushdown automata. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 145–157. Springer, Heidelberg (2005)
6. Chatterjee, K., de Alfaro, L., Henzinger, T.: The complexity of stochastic Rabin and Streett games. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 878–890. Springer, Heidelberg (2005)
7. Chatterjee, K., Jurdzinski, M., Henzinger, T.: Simple stochastic parity games. In: Meinke, K., Börger, E., Gurevich, Y. (eds.) CSL 1993. LNCS, vol. 832, pp. 100–113. Springer, Heidelberg (1994)
8. Chatterjee, K., Jurdzinski, M., Henzinger, T.: Quantitative stochastic parity games. In: Proceedings of SODA 2004, SIAM, pp. 121–130 (2004)
9. Feinberg, E., Shwartz, A. (eds.): Handbook of Markov Decision Processes. Kluwer, Dordrecht (2002)

10. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6, 512–535 (1994)
11. Kučera, A., Stražovský, O.: On the controller synthesis for finite-state Markov decision processes. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 541–552. Springer, Heidelberg (2005)
12. Mahadevan, S.: Partially observable semi-Markov decision processes: Theory and applications in engineering and cognitive science. In: AAAI: Fall Symposium on Planning with Partially Observable Markov Decision Processes (1998)
13. Puterman, M.L.: Markov Decision Processes. Wiley, Chichester (1994)
14. Safra, S.: Complexity of automata on infinite objects. PhD thesis (1989)
15. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Banff Higher Order Workshop, pp. 238–266 (1995)

Timed Concurrent Game Structures[⋆]

Thomas Brihaye, François Laroussinie, Nicolas Markey, and Ghassan Oreiby[⋆⋆]

Laboratoire Spécification & Vérification – CNRS & ENS Cachan – France

Abstract. We propose a new model for timed games, based on concurrent game structures (CGSs). Compared to the classical *timed game automata* of Asarin *et al.* [8], our timed CGSs are "more concurrent", in the sense that they always allow all the agents to act on the system, independently of the delay they want to elapse before their action. Timed CGSs weaken the "element of surprise" of timed game automata reported by de Alfaro *et al.* [15].

We prove that our model has nice properties, in particular that model-checking timed CGSs against timed ATL is decidable *via* region abstraction, and in particular that strategies are "region-stable" if winning objectives are. We also propose a new extension of TATL, containing ATL*, which we call TALTL. We prove that model-checking this logic remains decidable on timed CGSs. Last, we explain how our algorithms can be adapted in order to rule out Zeno (co-)strategies, based on the ideas of Henzinger *et al.* [15,21].

1 Introduction

Verification and model-checking. Over the last 30 years, the crucial role of verification has been emphasized by the unprecedented development of automated and embedded systems in various domains such as automotive industry, avionics or mobile communications.

Model-checking [25] is a technique of formal, model-based verification. This technique consists in exhaustively and automatically checking that all the behaviours of the (model representing the) system are consistent with some given formal specification. It is classical to represent the system (e.g. a network of computers and printers) as a finite-state system (a.k.a. Kripke structure), and to express the specifications (e.g., that any message sent by some computer always reaches its addressee) in some temporal logic, such as LTL (linear-time temporal logic) or CTL (computation-tree logic) [17]. Several efficient model-checking tools have been developped and applied with great success over an abundant number of industrial case studies [24,13,22].

Since the early 90's, this setting has been lifted to real-time, in particular with the introduction of *timed automata* [2], the extension of temporal logics to include quantitative requirements [1,4], and the development of efficient algorithms and tools [20,14,9]. This area is now very mature and widely applied for industrial case studies.

⋆ Work partly supported by project *DOTS* (ANR-06-SETI-003).

⋆⋆ This author is supported by a PhD grant from Région Ile-de-France.

Control and game theory. Control theory [11,27] is another facet of formal, model-based verification, geared towards the analysis of *open* systems, interacting with an (hostile) environment. The ultimate goal of this technique is to automatically synthesize a *controller* that will restrict the behaviour of the system in order to satisfy some given property. This problem is often encoded as a game-theoretic problem: the game is played by several players on a board, e.g. a *concurrent game structure* (CGS) [6]. CGSs are finite-state automata whose evolution conforms to the following protocol: at each step, all the players select one of the moves they are allowed to play, and the next state is looked up in the transition table of the CGS.

Alternating-time temporal logic (ATL) [5] has been proposed as an extension of CTL with *strategy quantifiers*. It can express controllability properties, e.g. "A has a strategy to eventually get its request served". Compared to CTL, this extension comes with no extra cost: it can still be verified in polynomial time [6].

Timed games. Several research works have focused on extending games to the real-time world. In that view, the best-established model is that of timed game automata [23,8,15,12,21]. A timed game automaton (TGA) is a timed automaton whose set of transitions is partitionned amongst the different players. At each step, each player chooses one of her possible transitions, as well as some amount of time she would like to wait before firing her selected transition. The player with the smallest delay is "elected", and her choices are applied. In case several player draw a tie, one of them is elected non-deterministically (or there can be a hierarchy among the players, but this breaks symmetry).

The logic ATL has also been extended to TATL, involving formula clocks to express timing requirements. It is decidable in exponential time whether a TATL formula is fulfilled in a timed game automaton [21]. Moreover, it is possible to restrict to "fair" strategies, ruling out strategies that consist in preventing time to diverge (a.k.a. Zeno strategies) [15,21].

Our contributions. Timed game automata are more of a game extension of timed automata than a timed extension of game structures. In this paper, we propose a timed extension of CGSs, which we call TCGSs. In those games, each player still chooses a delay and a move she wants to play after that delay, but she also proposes a function telling which moves she wants to play if someone proposes a smaller delay. That way, even if an opponent chooses a smaller delay, her behavior can still be "restricted" by the other players. This also avoids resorting to non-determinism in case several players choose the same delay. This could be useful in our example of a communication network, where two messages sent at the same time would result in a collision.

We prove that our model has nice properties: the functions proposed by the players can be chosen to be region-based (i.e., they can be constant on each region), and that the satisfaction of TATL properties is stable by regions. This provides us with an EXPTIME algorithm for model-checking TATL on our TCGSs, which we prove can be extended to force the players to play "fairly" w.r.t. divergence of time.

We also propose a new (to the best of our knowledge) temporal logic TALTL, which contains TATL and ATL* (and can thus express e.g. fairness properties) while remaining decidable (in 2EXPTIME). As a side result, we obtain that model-checking the corresponding TCLTL logic (containing TCTL and LTL) on timed automata is decidable in EXPSPACE.

Related works. As already mentionned, several papers have already dealt with timed games, especially in the framework of timed game automata. In [28], Schobbens and Bontemps propose a model for multi-agent games (called "real-time concurrent game structures", but that is still different from our TCGSs), and describe an algorithm for model-checking a timed extension of ATL incorporating MITL [3]. Their algorithm relies on event-clock automata, and is very different to our approach. The complexity of the procedure is not discussed there.

Outline of the paper. The paper is organized as follows: in Section 2, we formally define our timed concurrent game structures. Section 3 is devoted to showing that strategies can be made region-based (for region-based objectives). Section 4 proves that region-equivalence is a correct abstraction, and Section 5 describes the model-checking algorithms for TATL and TALTL. Last, in Section 6, we explain how our results can be extended to rule out Zeno strategies. Due to lack of space, proofs are omitted. They are detailed in [10].

2 Definitions

2.1 Untimed Concurrent Game Structures

We briefly recall the definition of concurrent game structures, which are multi-agent extensions of transition systems [6]. We extend the original definition in not requiring them to be finite-state, as we will use them for defining the semantics of our timed game structures. In the whole paper, Σ is a fixed finite alphabet.

Definition 1. *A* concurrent game structure *(CGS for short) is a tuple*[1] $\mathcal{S} = \langle \mathsf{Q}, \mathsf{Q}_0, l, \delta, \mathsf{Agt}, \mathbb{M}, \mathsf{Mv}, \mathsf{Edg}\rangle$ *where:*

- $\langle \mathsf{Q}, \mathsf{Q}_0, l, \delta\rangle$ *is a transition system with* $l\colon \mathsf{Q} \to \Sigma$*,*
- $\mathsf{Agt} = \{a_1, ..., a_k\}$ *is a finite set of agents,*
- $\mathbb{M}$ *is the set of all possible moves of the agents,*
- $\mathsf{Mv}\colon Q \times \mathsf{Agt} \to \mathcal{P}(\mathbb{M}) \setminus \varnothing$ *defines the set of possible moves for each player,*
- $\mathsf{Edg}\colon Q \times \mathbb{M}^k \to \delta$ *is the transition table, assigning a transition to each set of moves of the agents in each state. We further demand that transitions given by* $\mathsf{Edg}(q, m_{a_1}, ..., m_{a_k})$ *depart from* q*.*

We write $\mathsf{Exec}_{\mathcal{S}}$ (resp. $\mathsf{Exec}^F_{\mathcal{S}}$) for the set of (resp. finite) executions or trajectories of $\mathcal{S}$ (i.e., of the underlying transition system). Let $r = (r_i)_{0\leqslant i\leqslant n} \in \mathsf{Exec}^F_{\mathcal{S}}$. The length $|r|$ of r is n, the last location $\mathsf{last}(r)$ of r is r_n and, for any $m \leqslant n$, the m-th prefix $r_{\leqslant m}$ of r is the finite execution $(r_i)_{0\leqslant i\leqslant m}$.

[1] We might omit to mention $\mathbb{M}$ when it is clear from the context.

In a CGS, the transitions to be fired are chosen concurrently by all the agents: in some location q, each agent a_l selects a move $m_l \in \mathsf{Mv}(q, a_l)$. The resulting transition is indicated by the value of $\mathsf{Edg}(q, a_1, ...a_k)$. We now formalize this behavior.

Definition 2. *Let $\mathcal{S} = \langle Q, Q_0, l, \delta, Agt, \mathbb{M}, Mv, Edg\rangle$ be a CGS, and $a \in Agt$ be an agent. A strategy for a is a mapping $\lambda\colon Exec^F_{\mathcal{S}} \to \mathbb{M}$ s.t., for any $r \in Exec^F_{\mathcal{S}}$, $\lambda(r) \in Mv(\mathsf{last}(r), a_l)$. Given a coalition $A \subseteq Agt$, a strategy for A is a family $\lambda_A = (\lambda_l)_{a_l \in A}$ of strategies, one for each agent in A.*

Given a location q and a set of moves $m_l \in \mathsf{Mv}(q, a_l)$ for some agents a_l of a coalition A, the set of possible transitions from q under choices $(m_l)_{a_l \in A}$ is defined as $\mathsf{Next}(q, (m_l)_{a_l \in A}) = \{\mathsf{Edg}(q, m'_1, ..., m'_l) \mid \forall a_l \in A.\ m'_l = m_l\}$. In the same way, given a finite trajectory r and a strategy $\lambda_A = (\lambda_l)_{a_l \in A}$, we define $\mathsf{Next}(r, \lambda_A) = \mathsf{Next}(\mathsf{last}(r), (\lambda_l(r))_{a_l \in A})$.

An outcome of strategy λ_A after a finite execution r of length n is an execution $r' = (r'_i)_i$ s.t. r is a prefix of r' and, for any m, $(r'_{n+m}, r'_{n+m+1}) \in \mathsf{Next}(r'_{\leqslant n+m}, \lambda_A)$. We write[2] $\mathsf{Out}_{\mathcal{S}}(r, \lambda_A)$ for the set of outcomes of λ_A after r.

The aim of a strategy is generally to win the game, i.e., to enforce that any (infinite) outcome belongs to a given set of *winning trajectories*. Such a set is called a *winning objective*.

2.2 Timed Concurrent Game Structures

Given a set $\mathcal{C}$ of clock variables, a clock valuation is a mapping $v\colon \mathcal{C} \to \mathbb{R}^+$. Given a valuation v, a delay $t \in \mathbb{R}^+$ and a subset $Z \subseteq \mathcal{C}$, the valuation $v' = v + t$ is defined by $v'(c) = v(c) + t$ for all $c \in \mathcal{C}$, and the valuation $v'' = v[Z \leftarrow 0]$ is defined by $v''(c) = v(c)$ if $c \notin Z$ and $v''(c) = 0$ otherwise. We write v_0 for the valuation s.t. $v_0(c) = 0$ for any $c \in \mathcal{C}$.

Let M be a positive integer. The set of *clock constraints bounded by M* is the set of formulas defined by the following grammar:

$$\mathsf{Constr}(\mathcal{C}, M) \ni \phi ::= c \sim n \mid \phi \wedge \phi$$

where c ranges over $\mathcal{C}$, $n \leqslant M$ is an integer, and $\sim \in \{<, \leqslant, =, \geqslant, >\}$. We write $\mathsf{Constr}(\mathcal{C})$ for the set of unbounded clock constraints (i.e., when $M = +\infty$). That a clock valuation satisfies a clock constraint is defined in the obvious way.

Definition 3. *A* timed automaton *(TA for short) [2] is a tuple $\mathcal{A} = \langle Q, Q_0, l, \mathcal{C}, Inv, \delta\rangle$ where:*

- *Q is a finite set of locations, those in Q_0 being initial;*
- *$l\colon Q \to \Sigma$ labels each location with one letter of the alphabet;*
- *$\mathcal{C}$ is a finite set of clock variables;*
- *$Inv\colon Q \to Constr(\mathcal{C})$ defines the invariants of each location;*

[2] We will omit to mention the subscript $\mathcal{S}$ when it is clear from the context.

– $\delta \subseteq \mathsf{Q} \times (\mathsf{Q} \times 2^{\mathcal{C}})^{(\mathbb{R}^+)^{\mathcal{C}}}$ *is a finite set of transitions, required to fulfill the following requirement: if* $(q, f) \in \delta$*, then* f *is total, and there exists a positive integer* M *s.t., if* v *and* v' *are two clock valuations satisfying exactly the same set of formulas in* $\mathsf{Constr}(\mathcal{C}, M)$*, then* $f(v) = f(v')$*.*

Note that our definition of a transition is unusual. In our setting, a transition is a (total) function that assigns a location and a set of clocks to be reset to each valuation of the clocks. While both definitions are expressively equivalent, our modified definition will facilitate the extension to games.

The semantics of TAs is defined in terms of an infinite (timed) transition system. Here, we combine a delay transition with an action transition:

Definition 4. *With a TA* $\mathcal{A} = \langle \mathsf{Q}, \mathsf{Q}_0, l, \mathcal{C}, \mathsf{Inv}, \delta\rangle$*, we associate an infinite timed transition system (TTS for short)* $\mathcal{S} = \langle \mathsf{S}, \mathsf{S}_0, l', R\rangle$ *defined as follows:*

– $\mathsf{S} = \{(q, v) \in \mathsf{Q} \times (\mathbb{R}^+)^{\mathcal{C}} \mid v \models \mathsf{Inv}(q)\}$*, whose elements are called the* states *of* $\mathcal{A}$*;*
– $\mathsf{S}_0 = \mathsf{S} \cap (\mathsf{Q}_0 \times \{v_0\})$*;*
– $l'((q, v)) = l(q)$*;*
– $R \subseteq \mathsf{S} \times \mathbb{R}^+ \times \mathsf{S}$ *is such that* $(s, d, s') \in R$ *iff, writing* $s = (q, v)$ *and* $s' = (q', v')$*, there exists a transition* $(q, f) \in \delta$ *and a subset* $Z \subseteq \mathcal{C}$ *s.t.*$(q, v + d) \in \mathsf{S}$*,* $(q', Z) = f(v + d)$*, and* $v' = v + d[Z \leftarrow 0]$*.*

A (continuous) *execution* ρ of $\mathcal{A}$ is a (finite or infinite) sequence $((s_i, d_i))_i$ s.t. $(s_i, d_i, s_{i+1}) \in R$ for any i. Given such an execution $\rho = ((s_i, d_i))_i$, a *position* along ρ is a pair $(k, d) \in \mathbb{N}\times\mathbb{R}^+$ where $0 \leqslant d \leqslant d_k$. Writing $s_i = (q_i, v_i)$ for each i, the position (k, d) represents the state $(q_k, v_k + d)$. The set of positions of an execution are ordered lexicographically. Given two positions (k, d) and (k', d') with $(k, d) \leqslant (k', d')$, we define $\mathrm{time}((k, d), (k', d'))$ to be the delay elapsed between those two positions, namely $(d_k - d) + \sum_{k<j<k'} d_j + d'$.

We are now in a position to define our timed concurrent game structures:

Definition 5. *A* timed concurrent game structure *(TCGS for short) is a tuple* $\mathcal{T} = \langle \mathsf{Q}, \mathsf{Q}_0, l, \mathcal{C}, \mathsf{Inv}, \delta, \mathsf{Agt}, \mathbb{M}, \mathsf{Mv}, \mathsf{Edg}\rangle$ *where*

– $\langle \mathsf{Q}, \mathsf{Q}_0, l, \mathcal{C}, \mathsf{Inv}, \delta\rangle$ *is a TA;*
– Agt*,* $\mathbb{M}$*,* Mv *and* Edg *have the same properties as in CGSs.*

In this paper, we focus on finite-state TCGS, where both Q and $\mathbb{M} \subseteq \mathbb{N}$ are finite. This restriction is implicit in the sequel.

In a TCGS, the agents do not only choose the discrete action they want to play, but also the (non negative real) delay they would like to let elapse before their action takes place, and the actions they would play if the transition were to be taken earlier. Formally:

Definition 6. *Let* $\mathcal{T} = \langle \mathsf{Q}, \mathsf{Q}_0, l, \mathcal{C}, \mathsf{Inv}, \delta, \mathsf{Agt}, \mathbb{M}, \mathsf{Mv}, \mathsf{Edg}\rangle$ *be a TCGS,* $q \in \mathsf{Q}$*, and* $v \in (\mathbb{R}^+)^{\mathcal{C}}$*. A* full move *of a player* $a \in \mathsf{Agt}$ *from location* q *under valuation* v *is a pair* (t, f) *where* $t \in \mathbb{R}^+$ *and* $f\colon \mathbb{R}^+ \to \mathsf{Mv}(q, a)$ *s.t.* $v + t \models \mathsf{Inv}(q)$*. We write* $\mathsf{FM}((q, v), a)$ *for the set of full moves of player* a *in location* q *under* v*. We have that* $\mathsf{FM}((q, v), a) = \{t \in \mathbb{R}^+ \mid v + t \models \mathsf{Inv}(q)\} \times (\mathsf{Mv}(q, a))^{\mathbb{R}^+}$*. We write* $\mathsf{FM}(\mathbb{R}^+, \mathbb{M})$ *for the set* $\mathbb{R}^+ \times \mathbb{M}^{\mathbb{R}^+}$ *of all possible full moves.*

The semantics of a TGCS can then be defined in terms of an infinite CGS:

Definition 7. *With a TCGS $\mathcal{T} = \langle Q, Q_0, l, \mathcal{C}, \mathit{Inv}, \delta, \mathit{Agt}, \mathbb{M}, \mathit{Mv}, \mathit{Edg}\rangle$, we associate the infinite CGS $\mathcal{S} = \langle S, S_0, l', R, \mathit{Agt}, \mathit{FM}(\mathbb{R}^+, \mathbb{M}), \mathit{Mv}', \mathit{Edg}'\rangle$ defined as follows:*

- *$\langle S, S_0, l', R\rangle$ is the TTS associated with the TA $\langle Q, Q_0, l, \mathcal{C}, \mathit{Inv}, \delta\rangle$;*
- *for all $(q, v) \in S$ and for all $a \in \mathit{Agt}$, $\mathit{Mv}'((q, v), a) = \mathit{FM}((q, v), a)$;*
- *$\mathit{Edg}'((q, v),((t_1, f_1), ...,(t_k, f_k)))$ is defined as follows: letting $t_0 = \min\{t_i \mid i \leqslant k\}$, $m_i = f_i(t_0)$ for each $i \leqslant k$, $(q, f) = \mathit{Edg}(q,(m_1, ..., m_k))$, and $f(v + t_0) = (q', Z)$, we have $\mathit{Edg}'((q, v),((t_1, f_1), ...,(t_k, f_k))) = ((q, v), t_0,(q', v + t_0[Z \leftarrow 0]))$.*

Example 8. Let us consider the 2-player game depicted on Figure 1. In that figure, transitions are marked e.g. with $\langle a, b\rangle$ to indicate that they correspond to Player 1 playing move a and Player 2 playing move b (moves not drawn are assumed to be self-loops). One can be convinced that Player 1 has a strategy in state $(q_2, x = 0)$ for always avoiding location q_4. A possible full move is depicted on Figure 2.

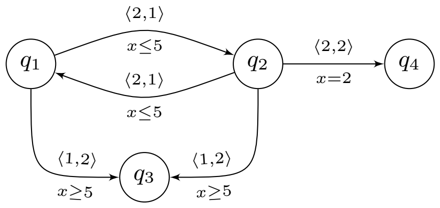


Fig. 1. Example of a TCGS

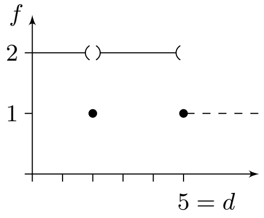


Fig. 2. A full move (d, f)

Remark 9. Even if we were not able to prove it formally, we think that our TCGSs and the classical model of TGAs [8,15] are incomparable w.r.t. alternating bisimilarity.

Still, TCGSs can be extended in many ways (e.g. with different invariants for each player, ...) while remaining decidable (with very little changes to our algorithms, and in particular with the same complexities). One possible extension is to have several transition tables, depending on the orders (and possible equalities) of the delays chosen by the different players. Such an extension would encompass TGAs (with an extra player for resolving non-determinism).

3 Region Equivalence and Strategies

3.1 Region Equivalence

Fix a family of integer constants M_x, one for each clock $x \in \mathcal{C}$. Two clock valuations v and v' are equivalent [2], denoted by $v \approx_t v'$, if, and only if, the following three conditions hold:

– for all $c \in \mathcal{C}$, either $\lfloor v(c) \rfloor = \lfloor v'(c) \rfloor$ or both $v(c)$ and $v'(c)$ are larger than M_c;
– for all $c, c' \in \mathcal{C}$ with $v(c) \leq M_c$ and $v(c') \leq M_{c'}$, $\mathrm{frac}(v(c)) \leq \mathrm{frac}(v(c'))$ if and only if $\mathrm{frac}(v'(c)) \leq \mathrm{frac}(v'(c'))$;
– for all $c \in \mathcal{C}$ with $v(c) \leq M_c$, $\mathrm{frac}(v(c)) = 0$ if and only if $\mathrm{frac}(v'(c)) = 0$.

The equivalence relation $\approx_t$ is extended to states and executions of a TA in the classical way. An equivalence class for $\approx_t$ on states is called a *region*. We write $[(q, v)]$ for the region containing (q, v), and $\mathcal{R}_\mathcal{A}$ for the set of all such regions. Since all the M_x are finite, the number of regions is finite.

Given a state (q, v) of a TA, the *timed future of* (q, v) $\mathrm{Fut}(q, v) \colon \mathbb{R}^+ \to \mathcal{R}_\mathcal{A}$ is defined as: $\mathrm{Fut}(q, v)(t) = [(q, v{+}t)]$. This function is piecewise constant and, since there are finitely many regions, $\mathbb{R}^+$ can be partitioned into finitely many intervals on which $\mathrm{Fut}(q, v)$ is constant. We write $\overline{\mathrm{Fut}}(q, v) = I_0\, I_1\, I_2\, \cdots\, I_l$ to denote that fact. In such a notation, $(I_i)_{i \leqslant l}$ is a sequence of intervals partitioning $\mathbb{R}^+$ and on each of which $\mathrm{Fut}(q, v)$ is constant. We also require that this list is minimal, i.e., for all $j \geqslant 0$ and any $t \in I_j$ and $t' \in I_{j+1}$, we have $\mathrm{Fut}(q, v)(t) \neq \mathrm{Fut}(q, v)(t')$.

We also define the immediate successor of a region as the partial function $\mathsf{succ} \colon \mathcal{R}_\mathcal{A} \to \mathcal{R}_\mathcal{A}$ defined by $\mathsf{succ}(r) = r'$ if $r' \neq r$ and there exists $(q, v) \in r$ and $t \in \mathbb{R}^+$ such that $(q, v + t) \in r'$ and $\forall 0 < t' < t$ we have $(q, v + t') \in r \cup r'$. We write succ^i for the i-th iterate of succ.

3.2 Simplifying Strategies

As is classical when dealing with timed systems, we will restrict to winning objectives that are region-definable, in the sense that a trajectory that is region-equivalent to a winning trajectory is also winning. Under this assumption, we prove that strategies can always be "region-definable". This is twofold: on the one hand, *region-invariance* means that the strategy does not depend on the whole history but only on its region abstraction; on the other hand, *region-uniformity* means that the value returned by the strategy is constant on the intermediate regions being visited. In order to define formally these notions we define the notion of isomorphism between two states. This notion will be the key tool in the proofs of existence of "region-definable" strategies.

Definition 10. *Let $\mathcal{A}$ be a TA and (q, v), (q', v') be two states of $\mathcal{A}$. We say that a bijection $\sigma \colon \mathbb{R}^+ \to \mathbb{R}^+$ is an* isomorphism *for (q, v) and (q', v') when:*

– *(increasing) for all t_1, $t_2 \in \mathbb{R}^+$, $t_1 < t_2$ iff $\sigma(t_1) < \sigma(t_2)$;*
– *(future-preserving) for all $t \in \mathbb{R}^+$, $\mathrm{Fut}(q, v)(t) = \mathrm{Fut}(q', v')(\sigma(t))$.*

Definition 11. *Let $\mathcal{T}$ be a TCGS, and λ_A be a strategy for a coalition $A \subseteq \mathsf{Agt}$.*

– *λ_A is* region-invariant *if, for all finite executions ρ and ρ' s.t. $\rho \approx_t \rho'$, there is an isomorphism σ for $\mathsf{last}(\rho)$ and $\mathsf{last}(\rho')$ s.t., writing $(d, f) = \lambda_A(\rho)$ and $(d', f') = \lambda_A(\rho')$, we have $d' = \sigma(d)$ and $f'(t) = f(\sigma^{-1}(t))$ for all $t \in \mathbb{R}^+$.*
– *λ_A is* region-uniform *if, for all finite execution ρ s.t. $\lambda_A(\rho) = (d, f)$, the value of f is constant on regions, i.e., writing $(q, v) = \mathsf{last}(\rho)$, for any $t, t' \in \mathbb{R}^+$, if $(q, v + t) \approx_t (q, v + t')$, then $f(t) = f(t')$.*

Roughly, region-invariance means that the strategy only depends on the projection of the history on regions, while region-uniformity means that the full-moves returned by the strategy are region-definable. In order to restrict to region-uniform and region-invariant strategies, we prove the following results [7]:

Proposition 12. *Let $\mathcal{T}$ be a TCGS and $A \subseteq \mathsf{Agt}$ be a coalition. For any two finite trajectories r and r' s.t. $r \approx_t r'$, for any strategy λ_A of coalition A, we can build a region-uniform and region-invariant strategy λ'_A s.t., for any $\rho' \in \mathsf{Out}(r', \lambda'_A)$, there exists $\rho \in \mathsf{Out}(r, \lambda_A)$ with $\rho \approx_t \rho'$.*

Corollary 13. *Let $\mathcal{T}$ be a TCGS, $A \subseteq \mathsf{Agt}$ be a coalition and Ω be a region-invariant winning objective. Let r and r' be two finite trajectories s.t. $r \approx_t r'$. There exists a winning strategy for A after r w.r.t. Ω if, and only if, there exists a* region-uniform and region-invariant *winning strategy for A after r' w.r.t Ω.*

4 Region CGS

In this section, for a TCGS $\mathcal{T}$, we define a finite CGS which we call the *region CGS* of $\mathcal{T}$. We show that this region CGS is *game-bisimilar* to the original TCGS. This region CGS is the "concurrent game version" of the classical region automaton [2]. Before we give the formal definition of the region CGS, we need to define the time-abstract version of a full move.

Definition 14. *Let $\mathcal{T} = \langle \mathsf{Q}, \mathsf{Q}_0, l, \mathcal{C}, \mathsf{Inv}, \delta, \mathsf{Agt}, \mathbb{M}, \mathsf{Mv}, \mathsf{Edg}\rangle$ be a TCGS. A* discrete full move *of a player $a \in \mathsf{Agt}$ from a region $[(q, v)]$ is a pair (d, f) where $d \in \mathbb{N}$ and $f \colon \mathbb{N} \to \mathsf{Mv}(q, a)$.*

We write $\mathsf{FM}([(q, v)], a)$ for the set of discrete full moves of player a in region $[(q, v)]$. We have that $\mathsf{FM}([(q, v)], a) = \left|\overline{\mathrm{Fut}}((q, v))\right| \times (\mathsf{Mv}(q, a))^{\mathbb{N}}$. The set $\mathsf{FM}(\mathbb{N}, \mathbb{M})$ denotes the set $\mathbb{N} \times \mathbb{M}^{\mathbb{N}}$ of all possible discrete full moves.

As can be expected, the first element of a full move specifies the delay that the agent would like to wait before firing her transition, in terms of the number of regions to be visited. The second item is the move function for the intermediary regions.

Definition 15. *With a TCGS $\mathcal{T} = \langle \mathsf{Q}, \mathsf{Q}_0, l, \mathcal{C}, \mathsf{Inv}, \delta, \mathsf{Agt}, \mathbb{M}, \mathsf{Mv}, \mathsf{Edg}\rangle$, we associate the (finite) region CGS $\mathcal{R} = \langle \mathsf{S}, \mathsf{S}_0, l', R, \mathsf{Agt}, \mathsf{FM}(\mathbb{N}, \mathbb{M}), \mathsf{Mv}', \mathsf{Edg}'\rangle$ with*

- *$\langle \mathsf{S}, \mathsf{S}_0, l', R\rangle$ is the region automaton associated with the TA $\langle \mathsf{Q}, \mathsf{Q}_0, l, \mathcal{C}, \mathsf{Inv}, \delta\rangle$;*
- *for all $s \in \mathsf{S}$, for all $a \in \mathsf{Agt}$, $\mathsf{Mv}'(s, a) = \mathsf{FM}(s, a)$;*
- *$\mathsf{Edg}'(s, ((d_1, f_1), ..., (d_k, f_k)))$ is defined as follows: let $d_0 = \min\{d_i \mid i \leqslant k\}$, $s' = \mathsf{succ}^{d_0}(s)$, $m_i = f_i(d_0)$ for each $i \leqslant k$, $(q, f) = \mathsf{Edg}(q, (m_1, ..., m_k))$, and $(q', Z) = f(q, v')$ for some $(q, v') \in s'$ (this does not depend on the choice of v' since f is definable with clock constraints); then $\mathsf{Edg}'(s, ((d_1, f_1), ..., (d_k, f_k))) = [(q', v'[Z \leftarrow 0])]$.*

Let $\mathcal{T}$ be a TGCS and $\mathcal{R}$ be its region CGS. With a run r of $\mathcal{T}$, one can naturally associate a unique run of $\mathcal{R}$, which we denote by $[r]$.

Using Prop 12, it can be proved that this abstraction is correct, in the sense that a TCGS $\mathcal{T}$ and its region abstraction $\mathcal{R}$ are game-bisimilar, i.e., that any strategy in one of those structures can be "mimicked" in the other one [7].

Theorem 16. *Let $\mathcal{T}$ be a TCGS. Region equivalence induces a game-bisimulation between (the infinite CGS associated with) $\mathcal{T}$ and its region CGS $\mathcal{R}$.*

5 Timed ATL

We will study several extensions of the logic ATL defined originally in [5]. We begin with defining the largest extension, namely TATL*.

Definition 17. *The logic TATL* is defined by the following grammar:*

$$\begin{aligned} \mathsf{TATL}^* \ni \phi_s &::= p \mid c \sim n \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \langle\!\langle A \rangle\!\rangle\, \phi_p \\ \phi_p &::= \phi_s \mid c.\phi_p \mid \phi_p \wedge \phi_p \mid \phi_p \vee \phi_p \mid \phi_p \,\mathbf{U}\, \phi_p \mid \phi_p \,\mathbf{R}\, \phi_p \end{aligned}$$

where p ranges over Σ, c ranges over a finite set of formula clocks, $\sim \in \{<, \leqslant, =, \geqslant, >\}$, $n \in \mathbb{N}$, and $A \subseteq \mathsf{Agt}$. Formulas of the form ϕ_s are called state formulas*, while formulas of the form ϕ_p are* path formulas.

A TATL* formula ϕ is interpreted over a position p along a trajectory r of a TCGS $\mathcal{T}$ w.r.t. a valuation w for formula clocks. The semantics of $\mathcal{T}, r, p \models_w \phi$ is defined in the usual way for atomic propositions, clock comparisons and the freeze quantifier "$c.\psi$" [4], boolean combinators, and (dual) modalities $\mathbf{U}$ and $\mathbf{R}$. As usual, we use $\mathbf{F}\,\phi$ as a shorthand for $\top\,\mathbf{U}\,\phi$ (eventually ϕ), $\mathbf{G}\,\phi$ for $\perp\,\mathbf{R}\,\phi$ (always ϕ), $\overset{\infty}{\mathbf{F}}\,\phi$ for $\mathbf{G}\,\mathbf{F}\,\phi$, and $\overset{\infty}{\mathbf{G}}\,\phi$ for $\mathbf{F}\,\mathbf{G}\,\phi$. The strategy quantifier $\langle\!\langle A \rangle\!\rangle\,\psi_p$ expresses the existence of a strategy for coalition A all of whose outcomes satisfy ψ_p [5]. The truth value of a state formula ϕ_s only depends on the current position and the trajectory can then be omitted in that case. We write $\mathcal{T}, (q, v) \models_w \phi_s$ when $\mathcal{T}, r, (0, 0) \models_w \phi_s$ for some trajectory starting in (q, v).

There is no hope of being able to verify TATL* as this logic is a superset of TPTL, which is known to be undecidable on timed automata under our continuous semantics [4]. We thus focus on fragments of TATL* (that inherit their semantics from the above semantics of TATL*). We define ATL* as being the (classical) fragment of TATL* not involving clocks, and TATL and ATL as the fragments of TATL* and ATL*, resp., in which path formulas restricted to the following grammar:

$$\phi_p ::= \phi_s \,\mathbf{U}\, \phi_s \mid \phi_s \,\mathbf{R}\, \phi_s \mid c.\phi_p.$$

We also define a new fragment of TATL*, which we call TALTL, containing both ATL* and TATL:

Definition 18. *The syntax of* TALTL *is defined by the following grammar:*

$$\mathsf{TALTL} \ni \phi_s ::= p \mid c \sim n \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \langle\!\langle A \rangle\!\rangle\, \phi_p \mid c.\phi_s$$
$$\phi_p ::= \phi_s \mid \phi_p \wedge \phi_p \mid \phi_p \vee \phi_p \mid \phi_p \,\mathbf{U}\, \phi_p \mid \phi_p \,\mathbf{R}\, \phi_p$$

Remark 19. To our knowledge, TALTL has never been studied earlier, even in the setting of timed automata. The difference with TATL* lies in the fact that clocks can now only be reset in state formulas, and not in path formulas. We believe that our new intermediate logic is really interesting for model-checking (despite is rather high complexity). Indeed, it extends timed branching-time logics with e.g. fairness (for instance, $c.\,\mathbf{A}(\overset{\infty}{\mathbf{F}}\, p \Rightarrow \mathbf{F}\,(q \wedge \mathbf{F}\,(q' \wedge c \leqslant 10)))$ is a TCLTL formula, stating that along fair executions, q and then q' will occur within 10 time units) while remaining decidable[3].

We begin with proving that TALTL cannot distinguish between two region-equivalent states of a TCGS. This requires to extend our previous definitions: given a TCGS $\mathcal{T} = \langle \mathsf{Q}, \mathsf{Q}_0, l, \mathcal{C}, \mathsf{Inv}, \delta, \mathsf{Agt}, \mathsf{Mv}, \mathsf{Edg}\rangle$ and a set of formula clocks $\mathcal{C}'$ (disjoint from $\mathcal{C}$), we define $\mathcal{T}' = \langle \mathsf{Q}, \mathsf{Q}_0, l, \mathcal{C} \cup \mathcal{C}', \mathsf{Inv}, \delta, \mathsf{Agt}, \mathsf{Mv}, \mathsf{Edg}\rangle$. This TCGS involves $\mathcal{C}'$, but its clocks do not play any role in the semantics. In such an extended TCGS, we write (q, v, w) for a state of the infinite CGS associated with $\mathcal{T}'$, where v is a valuation of clocks in $\mathcal{C}$ and w is a valuation of the clocks in $\mathcal{C}'$. We write $\mathsf{proj}((q, v, w)) = (q, v)$, and extend this notation to map trajectories in $\mathcal{T}'$ to their corresponding trajectory in $\mathcal{T}$.

Theorem 20. *Let $\mathcal{T}$ be a TCGS, and $\mathcal{T}'$ be the corresponding TCGS extended with a set of formula clocks $\mathcal{C}'$. Let ϕ be a (path- or state-) TALTL formula built on the clocks in $\mathcal{C}'$. For any region-equivalent states (q, v, w) and (q, v', w') of $\mathcal{T}'$, and any two region-equivalent trajectories r and r' starting from (q, v, w) and (q, v', w'), resp., we have*

$$\mathcal{T}, \mathit{proj}(r), (0,0) \models_w \phi \quad \text{iff} \quad \mathcal{T}, \mathit{proj}(r'), (0,0) \models_{w'} \phi.$$

Moreover, if ϕ is a state formula, then this result holds even if we relax the hypothesis that r and r' be region-equivalent.

Remark 21. It should be noticed that the above result fails to hold for TATL*: it is easy to find an example of two trajectories visiting the same sequence of extended regions, but only one of which satisfies formula $c.\mathbf{F}\,(q \wedge c \geqslant 1)$. We don't know if the result of Theorem 20 holds for *state-formulas* of TATL*.

5.1 Model-Checking

We now describe a region-based algorithm for model-checking TCGSs against TALTL. Given a TCGS $\mathcal{T}$ and a set of formula clocks $\mathcal{C}'$, we consider the TCGS $\mathcal{T}'$ extending $\mathcal{T}$ with the clocks in $\mathcal{C}'$, and write $\mathcal{R}'$ for the associated region CGS.

[3] From our results below, it is easy to convince that model-checking TCLTL (i.e., TALTL with path quantifiers instead of strategy quantifiers) is in EXPSPACE.

Our algorithm labels this finite-state CGS with state-subformulas of a given TALTL state-formula ψ to be checked on $\mathcal{T}$. This is achieved by recursively filling a boolean table $T([(q,v,w)],\psi)$, where $[(q,v,w)]$ ranges over the set of regions of $\mathcal{R}'$ and ψ ranges over the state-subformulas of ϕ. Our algorithm uses an extra procedure `ATLstar-labeling`, which is the classical algorithm for ATL* model-checking, as defined in [6].

Theorem 22. *Let $\mathcal{T}$ be a TCGS, $\mathcal{C}'$ be a set of formula clocks, $\mathcal{T}'$ be the extended TCGS with clocks of $\mathcal{C}'$, and $\mathcal{R}'$ be the region CGS corresponding to $\mathcal{T}'$. Let $\phi \in$ TALTL built on formula clocks in $\mathcal{C}'$. Let T be the table obtained after applying the algorithm described above on $\mathcal{R}'$ and ϕ. Let (q,v,w) be a state in $\mathcal{T}'$. Then $\mathcal{T},(q,v) \models_w \phi$ iff $T([(q,v,w)],\phi) = \top$.*

Of course, a more efficient algorithm is obtained for TATL by replacing `ATLstar-labeling` by the PTIME procedure `ATL-labeling` for ATL formulas. As a corollary, we obtain the following theorem:

Theorem 23. *Model-checking TALTL on TCGSs is decidable and 2EXPTIME-complete. Model-checking TATL on TCGSs is EXPTIME-complete.*

Remark 24. Strategies for ATL can always be chosen memoryless. This however does not extend to TATL, since the region CGS contains extra information about formula clocks, which are not part of the model.

6 Ruling Out Zeno Strategies

In the previous section, we proved the decidability of the TALTL model-checking problem with no restriction on the strategies. In particular, a player could achieve a safety objective by blocking time. In this section, in order to forbid this kind of unrealistic behaviors, we explain how we can force the players to play "fairly", ruling out strategies that consist in preventing time to diverge (a.k.a. Zeno strategies). Zenoness is the fact for an infinite execution to be time-convergent: an infinite execution $((s_i,d_i))_{i\in\mathbb{N}}$ is Zeno if $\sum_{i\in\mathbb{N}} d_i < \infty$. To forbid Zeno strategies, we use the framework introduced in [15]. Let us mention that a similar setting appears in [18] in order to handle Zeno executions of *Timed I/O Automata*.

Following [15], we begin with detecting Zeno outcomes. In order to do so, given a TCGS $\mathcal{T}$, we add to it an extra clock $z \notin \mathcal{C}$ which is reset when it exceeds 1. This is formally achieved as follows: Let $\mathcal{T} = \langle \mathsf{Q}, \mathsf{Q}_0, l, \mathcal{C}, \mathsf{Inv}, \delta, \mathsf{Agt}, \mathsf{Mv}, \mathsf{Edg}\rangle$ be a TCGS. We define $\mathcal{T}_z = \langle \mathsf{Q}, \mathsf{Q}_0, l, \mathcal{C}_z, \mathsf{Inv}, \delta_z, \mathsf{Agt}, \mathsf{Mv}, \mathsf{Edg}'\rangle$ where $\mathcal{C}_z = \mathcal{C} \cup \{z\}$ and δ_z is defined from δ as follows: for each $(q,f) \in \delta$, we have $(q,f') \in \delta'$ where $f' \colon (\mathbb{R}^+)^{\mathcal{C}_z} \to Q \times 2^{\mathcal{C}_z}$ is defined as follows:

$$f'(x_1,\ldots,x_n,z) = \begin{cases} (q',Z) & \text{if } f'(x_1,\ldots,x_n) = (q',Z) \text{ and } z < 1 \\ (q',Z\cup\{z\}) & \text{if } f'(x_1,\ldots,x_n) = (q',Z) \text{ and } z \geq 1. \end{cases}$$

The transition table Edg' then is adapted to δ_z in the obvious way. Given an infinite executions r of $\mathcal{T}'$, we clearly have that r is a non-Zeno execution if and only if clock z is reset infinitely often.

When an execution is Zeno, it might be the case that only part of the players are responsible for it. A player is responsible for the Zenoness of an execution if she is "elected" infinitely many times for her choosing the smallest delay. A coalition is responsible for Zenoness if at least one of its member is. To record that information, we decorate the infinite CGS associated to a given TCGS $\mathcal{T}$ with "blames". This requires to extend the alphabet Σ to $\Sigma' = \Sigma \times \{\mathsf{tick}, \overline{\mathsf{tick}}\} \times \{\mathsf{bl}_A, \mathsf{bl}_{\bar{A}}, \mathsf{bl}_{\mathsf{Agt}}\}$. The first two symbols will be used when clock z reaches 1, while the last three assign a blame to either coalition A, their opponent $\bar{A}$, or to both. This is the underlying alphabet in the extended CGS $\mathcal{E}_A$ defined below:

Definition 25. *Let $\mathcal{T} = \langle Q, Q_0, l, \mathcal{C}_z, Inv, \delta, Agt, Mv, Edg\rangle$ be a TCGS (assumed to be already extended with a "tick"-clock z), $\mathcal{S} = \langle S, S_0, l', R, Agt, Mv', Edg'\rangle$ be the associated infinite CGS, and $A \subseteq Agt$ be a coalition. We define the extended CGS $\mathcal{E}_A = \langle S^{\mathcal{E}}, S_0^{\mathcal{E}}, l^{\mathcal{E}}, R^{\mathcal{E}}, Agt, Mv^{\mathcal{E}}, Edg^{\mathcal{E}}\rangle$ as follows:*

- *$S^{\mathcal{E}} \subseteq S \times \{\mathsf{tick}, \overline{\mathsf{tick}}\} \times \{bl_A, bl_{\bar{A}}, bl_{Agt}\}$ and $S_0^{\mathcal{E}} = S_0 \times \{\overline{\mathsf{tick}}\} \times \{bl_{Agt}\}$ are extensions of S and S_0 with some extra informations for keeping track of whose choice has been considered;*
- *$l^{\mathcal{E}}((s,t,b)) = (l'(s), t, b)$,*
- *$R^{\mathcal{E}} \subseteq S^{\mathcal{E}} \times \mathbb{R}^+ \times S^{\mathcal{E}}$ contains two kinds of transitions:*
    - *for each $(s,t,b) \in S^{\mathcal{E}}$, with $s = (q,v)$, and $d \in \mathbb{R}^+$ s.t. $v(z) + d < 1$, then for each $(s,d,s') \in R$ and $b' \in \{bl_A, bl_{\bar{A}}, bl_{Agt}\}$, the transition $((s,t,b), d, (s', \overline{\mathsf{tick}}, b'))$ is in $R^{\mathcal{E}}$;*
    - *for each $(s,t,b) \in S^{\mathcal{E}}$, with $s = (q,v)$, and $d \in \mathbb{R}^+$ s.t. $v(z) + d \geqslant 1$, then for each $(s,d,s') \in R$ and $b' \in \{bl_A, bl_{\bar{A}}, bl_{Agt}\}$, the transition $((s,t,b), d, (s', \mathsf{tick}, b'))$ is in $R^{\mathcal{E}}$;*
- *$Edg^{\mathcal{E}}$ is defined from Edg' as follows: given the set of full moves $((d_l, f_l)_{a_l \in Agt}$, we let $d_0 = \min\{d_l \mid a_l \in Agt\}$ as before, and set bl to be:*
    - *bl_A if $\exists a_l \in A.\ d_l = d_0$ and $\forall a_l \in \bar{A}.\ d_l > d_0$;*
    - *$bl_{\bar{A}}$ if $\exists a_l \in \bar{A}.\ d_l = d_0$ and $\forall a_l \in A.\ d_l > d_0$;*
    - *bl_{Agt} if $\exists a_l \in A.\ d_l = d_0$ and $\exists a_l \in \bar{A}.\ d_l = d_0$;*

  Then, if $Edg'(s, ((d_1, f_1), ..., (d_k, f_k))) = (s, d_0, s')$, with $s = (q,v)$, we let

$$Edg^{\mathcal{E}}((s,t,b), ((d_1, f_1), ..., (d_k, f_k))) = ((s,t,b), d_0, (s', \overline{\mathsf{tick}}, bl))$$

  if $v(z) + d_0 < 1$, and for the other cases,

$$Edg^{\mathcal{E}}((s,t,b), ((d_1, f_1), ..., (d_k, f_k))) = ((s,t,b), d_0, (s', \mathsf{tick}, bl)).$$

It should be noticed that this infinite CGS does *not* correspond to an infinite CGS associated with a TCGS: it is generally not possible to decorate a TCGS with the informations about the players to blame. The clock-equivalence is naturally extended to the state of $\mathcal{E}_A$. We say that $(s,t,b) \approx_t (s',t',b')$ if and only if $s \approx_t s'$, $t = t'$ and $b = b'$. We keep the terminology of *region* for the equivalence class of an extended state (s,z,t,b) for $\approx_t$. This equivalence relation can also be extended to executions of $\mathcal{E}_A$. The definitions of region-invariant and region-uniform strategy naturally extend to this context.

Now, for a strategy of coalition A to be winning *without Zenoness*, all of its outcomes must either be non-Zeno and winning, or be Zeno and blame agents in A only finitely many times. We then obtain a theorem similar to Corollary 13:

Theorem 26. *Let $\mathcal{T}$ be a TCGS, $A \subseteq \mathsf{Agt}$ be a coalition and Ω be a region-invariant winning objective. Let r and r' be two isomorphic finite trajectories. There exists a winning non-Zeno strategy for A after r w.r.t. Ω if, and only if, there exists a* region-uniform and region-invariant *winning strategy without Zenoness for A after r' w.r.t Ω.*

Note that making a strategy region-uniform modifies the blames in the outcomes: only a weaker version of Prop. 12 holds in this case, but it is sufficient for our purpose.

In order to model-check TALTL formulas, we define a modified semantics that captures the notion of "winning without Zenoness":

Definition 27. *Let $\mathcal{T} = \langle Q, Q_0, \Sigma, l, \mathcal{C}, \mathit{Inv}, \delta, \mathit{Agt}, \mathbb{M}, \mathit{Mv}, \mathit{Edg}\rangle$ be a TCGS, and, for each coalition $A \subseteq \mathit{Agt}$, $\mathcal{E}_A = \langle S, S_0, l', R, \mathit{Agt}, \mathit{FM}(\mathbb{R}^+, \mathbb{M}), \mathit{Mv}', \mathit{Edg}'\rangle$ be the extended infinite CGS built in Definition 25. Let $s = (q, v)$ be a state of $\mathcal{S}$, and $w \colon \mathcal{C}' \to \mathbb{R}^+$ be a valuation of the formula clocks. Let $\phi \in \mathsf{TATL}^*$. That $\mathcal{T}, s \models^Z_w \phi$ is defined in the same way as $\mathcal{T}, s \models_w \phi$, except for $\phi = \langle\!\langle A \rangle\!\rangle\, \psi_p$:*

$$\mathcal{T}, s \models^Z_w \langle\!\langle A \rangle\!\rangle\, \psi_p \;\Leftrightarrow$$

$$\mathcal{E}_A, (s, \overline{\mathsf{tick}}, \mathit{bl}_A) \models_w \langle\!\langle A \rangle\!\rangle\, ((\overset{\infty}{\mathbf{F}}\, \neg \mathit{bl}_{\bar{A}} \Rightarrow \overset{\infty}{\mathbf{F}}\, \mathsf{tick}) \wedge (\overset{\infty}{\mathbf{F}}\, \mathsf{tick} \Rightarrow \psi_p))$$

Our first result is that TALTL under this semantics still cannot distinguish between two region-equivalent states. Note that, though the proof is similar, this result is not an immediate consequence of Theorem 20 as it is not possible to directly decorate TCGSs with blames.

Theorem 28. *Let $\mathcal{T}$ be a TCGS, and $\mathcal{T}'$ be the corresponding TCGS extended with a set of formula clocks $\mathcal{C}'$. Let ϕ be a TALTL formula built on the clocks in $\mathcal{C}'$. For any region-equivalent states (q, v, w) and (q, v', w') of $\mathcal{T}'$, and any two region-equivalent trajectories r and r' starting from (q, v, w) and (q, v', w'), resp., we have*

$$\mathcal{T}, \mathit{proj}(r), (0, 0) \models^Z_w \phi \quad \textit{iff} \quad \mathcal{T}, \mathit{proj}(r'), (0, 0) \models^Z_{w'} \phi.$$

Moreover, if ϕ is a state-formula, then this result holds even if we relax the assumption that r and r' be region-equivalent.

It is then possible to extend the region CGS to include also the informations about ticks and blames, and to adapt the algorithm for verifying the non-Zeno semantics. Again, the correctness of the algorithm is not straightforward, as region-equivalence is sometimes too coarse to keep precise informations about blames. In the case of TALTL, the algorithm will still rely on `ATLstar-labeling`, while for TATL, the state-formulas to verify will be in FairATL with one strong-fairness constraint [6]. It should be noted that blames in the extended region CGS do not always correspond to blames in the TCGS extended with formulas clocks. In the end:

Theorem 29. *Under the non-Zenoness semantics, model-checking TALTL on TCGSs is decidable and 2*EXPTIME*-complete, model-checking TATL on TCGSs is* EXPTIME*-complete.*

7 Conclusion and Perspectives

We have proposed a new model for timed games, by extending concurrent game structures of [6] to the real-time setting. We proved that our model is compatible with region abstraction, and that TATL can be model-checked in EXPTIME (because of the binary encoding of the constants in the automaton and in the formula), matching the complexity obtained in [21] for timed game automata. We also proposed an extension of TATL that also embeds ATL*, at the price of an extra exponential blowup for model-checking.

Table 1. Complexities of model-checking different logics on TCGSs

	algo. compl. w.r.t. ϕ and $\mathcal{T}$	theoretical complexity
ATL*	$2^{2^{O(\|\phi\|)}} \cdot 2^{O(\|\mathcal{T}\|)}$	2EXPTIME-complete
TATL	$2^{O(\|\phi\|)\cdot O(\|\mathcal{T}\|)}$	EXPTIME-complete
TALTL	$2^{2^{O(\|\phi\|)}} \cdot 2^{O(\|\mathcal{T}\|)}$	2EXPTIME-complete

As a future work, we plan to investigate synthesis of strategies. From our results, we already know that restriction to region-based strategies will be sufficient. The recent works of Harding *et al.* [19] could be a source of inspiration for this direction of research. Beyond non-Zenoness, we also would like to study robustness issues in timed games [26,16], this notion allows to distinguish inifinitely quick or precise strategies (that cannot be implemented over a real computer). Thus it would be interesting to decide the existence of robust strategies.

References

1. Alur, R., Courcoubetis, C., Dill, D.: Model-checking in dense real-time. Inf. & Comp. 104(1), 2–34 (1993)
2. Alur, R., Dill, D.: A theory of timed automata. TCS 126(2), 183–235 (1994)
3. Alur, R., Feder, T., Henzinger, T.: The benefits of relaxing punctuality. J. ACM 43(1), 116–146 (1996)
4. Alur, R., Henzinger, T.: A really temporal logic. J. ACM 41(1), 181–204 (1994)
5. Alur, R., Henzinger, T., Kupferman, O.: Alternating-time temporal logic. In: FOCS'97, pp. 100–109. IEEE Computer Society Press, Los Alamitos (1997)
6. Alur, R., Henzinger, T., Kupferman, O.: Alternating-time temporal logic. J. ACM 49(5), 672–713 (2002)
7. Alur, R., Henzinger, T., Kupferman, O., Vardi, M.: Alternating refinement relations. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 163–178. Springer, Heidelberg (1998)
8. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: Proc. Symp. System Structure & Control, pp. 469–474. Elsevier, Amsterdam (1998)

9. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
10. Brihaye, Th., Laroussinie, F., Markey, N., Oreiby, G.: Timed concurrent game structures. Technical report, LSV, ENS Cachan, France (2007)
11. Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Trans. AMS 138, 295–311 (1969)
12. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 66–80. Springer, Heidelberg (2005)
13. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: A new symbolic model verifier. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 495–499. Springer, Heidelberg (1999)
14. Daws, C., Olivero, A., Tripakis, S., Yovine, S.: The tool Kronos. In: Alur, R., Sontag, E.D., Henzinger, T. (eds.) Hybrid Systems III. LNCS, vol. 1066, pp. 208–219. Springer, Heidelberg (1996)
15. de Alfaro, L., Faella, M., Henzinger, T., Majumdar, R., Stoelinga, M.: The element of surprise in timed games. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 144–158. Springer, Heidelberg (2003)
16. De Wulf, M., Doyen, L., Markey, N., Raskin, J.-F.: Robustness and implementability of timed automata. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 118–133. Springer, Heidelberg (2004)
17. Emerson, E.: Temporal and modal logic. In: Handbook of Theoretical Computer Science, Formal Models and Sematics, vol. B, pp. 995–1072. Elsevier, Amsterdam (1990)
18. Gawlick, R., Segala, R., Søgaard-Andersen, J., Lynch, N.: Liveness in timed and untimed systems. In: Shamir, E., Abiteboul, S. (eds.) ICALP 1994. LNCS, vol. 820, pp. 166–177. Springer, Heidelberg (1994)
19. Harding, A., Ryan, M., Schobbens, P.-Y.: A new algorithm for strategy synthesis in LTL games. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 477–492. Springer, Heidelberg (2005)
20. Henzinger, T., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real time systems. Inf. & Comp. 111(2), 193–244 (1994)
21. Henzinger, T., Prabhu, V.: Timed alternating-time temporal logic. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 1–17. Springer, Heidelberg (2006)
22. Holzmann, G.: The model checker spin. IEEE Trans. Software Engineering 23(5), 279–295 (1997)
23. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems. In: Mayr, E.W., Puech, C. (eds.) STACS 95. LNCS, vol. 900, pp. 229–242. Springer, Heidelberg (1995)
24. McMillan, K.: Symbolic Model Checking — An Approach to the State Explosion Problem. PhD thesis, CMU, Pittsburgh, Pennsylvania, USA (1993)
25. Pnueli, A.: The temporal logic of programs. In: FOCS'77, pp. 46–57. IEEE Computer Society Press, Los Alamitos (1977)
26. Puri, A.: Dynamical properties of timed automata. In: Ravn, A.P., Rischel, H. (eds.) FTRTFT 1998. LNCS, vol. 1486, pp. 210–227. Springer, Heidelberg (1998)
27. Ramadge, P., Wonham, W.: The control of discrete event systems. Proc. IEEE 77(1), 81–98 (1989)
28. Schobbens, P.-Y., Bontemps, Y.: Real-time concurrent game structures. Personnal communication (2005)

Pushdown Module Checking with Imperfect Information[⋆]

Benjamin Aminof[1], Aniello Murano[2], and Moshe Y. Vardi[3]

[1] Hebrew University, Jerusalem 91904, Israel
[2] Università degli Studi di Napoli "Federico II", 80126 Napoli, Italy
[3] Rice University, Houston, TX 77251-1892, U.S.A

Abstract. The model checking problem for finite-state open systems (*module checking*) has been extensively studied in the literature, both in the context of environments with perfect and imperfect information about the system. Recently, the perfect information case has been extended to infinite-state systems (*pushdown module checking*). In this paper, we extend pushdown module checking to the imperfect information setting; i.e., the environment has only a partial view of the system's control states and pushdown store content. We study the complexity of this problem with respect to the branching-time temporal logic *CTL*, and show that pushdown module checking, which is by itself harder than pushdown model checking, becomes undecidable when the environment has imperfect information. We also show that undecidability relies on hiding information about the pushdown store. Indeed, we prove that with imperfect information about the control states, but a visible pushdown store, the problem is decidable and its complexity is the same as that of (perfect information) pushdown module checking.

1 Introduction

In system modeling we distinguish between *closed* and *open* systems [HP85]. In a closed system all the nondeterministic choices are internal, and resolved by the system. In an open system there are also external nondeterministic choices, which are resolved by the environment [Hoa85]. In order to check whether a closed system satisfies a required property, we translate the system into some formal model, specify the property with a temporal-logic formula, and check formally that the model satisfies the formula. Hence, the name *model checking* for the verification methods derived from this viewpoint ([CE81, QS81]).

In [KV96, KVW01], Kupferman, Vardi, and Wolper studied open finite-state systems. In their framework, the open finite-state system is described by a labeled state-transition graph called *module*, whose set of states is partitioned into a set of *system states* (where the system makes a transition) and a set of *environment states* (where the environment makes a transition). Given a module $\mathcal{M}$

[⋆] Work supported in part by MIUR FIRB Project no. RBAU1P5SS, NSF grants CCR-9988322, CCR-0124077, CCR-0311326, CCF-0613889, and ANI-0216467, by BSF grant 9800096, and by gift from Intel.

describing the system to be verified, and a temporal logic formula φ specifying the desired behavior of the system, the problem of model checking a module, called *module checking*, asks whether for all possible environments $\mathcal{M}$ satisfies φ. In particular, it might be that the environment does not enable all the external nondeterministic choices. Module checking thus involves not only checking that the full computation tree $\langle T_M, V_M \rangle$ obtained by unwinding $\mathcal{M}$ (which corresponds to the interaction of $\mathcal{M}$ with a maximal environment) satisfies the specification φ, but also that every tree obtained from it by pruning children of environment nodes (this corresponds to the different choices of different environments) satisfy φ. For example, consider an ATM machine that allows customers to deposit money, withdraw money, check balance, etc. The machine is an open system and an environment for it is a subset of the set of all possible infinite lines of customers, each with its own plans. Accordingly, there are many different possible environments to consider. It is shown in [KV96, KVW01] that for formulas in branching time temporal logics, module checking open finite-state systems is exponentially harder than model checking closed finite-state systems.

In [KV97] module checking has been extended to a setting where the environment has *imperfect information*[1] about the state of the system (see also [CH05, CDHR06], for related work regarding imperfect information). In this setting, every state of the module is a composition of *visible* and *invisible* variables, where the latter are hidden from the environment. While a composition of a module $\mathcal{M}$ with an environment with perfect information corresponds to arbitrary disabling of transitions in $\mathcal{M}$, the composition of $\mathcal{M}$ with an environment with imperfect information is such that whenever two computations of the system differ only in the values of internal variables along them, the disabling of transitions along them coincide. For example, in the above ATM machine, a person does not know, before he asks for money, whether or not the ATM has run out of paper for printing receipts. Thus, the possible behaviors of the environment are independent of this missing information. Given an open system $\mathcal{M}$ with a partition of $\mathcal{M}$'s variables into visible and invisible, and a temporal logic formula φ, the module-checking problem with imperfect information asks whether φ is satisfied by all trees obtained by pruning children of environment nodes from $\langle T_M, V_M \rangle$, according to environments whose nondeterministic choices are independent of the invisible variables. One of the results shown in [KV97] is that *CTL* module checking with imperfect information is EXPTIME-complete.

In recent years, model checking of pushdown systems has received a lot of attention (see for example [Wal96, Wal00, BEM97, EKS03]), largely due to the ability of pushdown systems to capture the flow of procedure calls and returns in programs [ABE$^+$05]. Recently, [BMP05] extended these techniques by introducing open pushdown systems (with perfect information) that interact with their environment. It is shown in [BMP05] that *CTL pushdown module checking* is 2EXPTIME-complete and thus much harder than pushdown model checking.

[1] In the literature, the term *incomplete information* is sometimes used to refer to what we call imperfect information.

Consider again the example of the ATM machine, where the information regarding the presence of printing paper is invisible to the customers. Suppose also that the ATM machine shows advertisements, and that it works under the constraint that the number of advertisements the customer must view, before the card can be taken out of the machine, is equal to the number of operations the customer performed. The described machine can be modeled as an open pushdown system $\mathcal{M}$ where control states take care of the operation performed by the ATM (interacting with customers), and the pushdown store is used to keep track of the advertisements that remain to be shown. Now, suppose that we want to verify that in all possible environments, it is always possible for an inserted card to be ejected. This requirement can be modeled by the CTL formula $\varphi = AG(\textbf{insert-card} \rightarrow EF\textbf{eject-card})$. Since the presence of printing paper is invisible to the customers, we have imperfect information about the control states of the module. If we allow the ATM to push, after each operation the customer makes, an invisible number (possibly zero) of pending advertisements, then we also have invisible information in the pushdown store.

In this paper, we extend pushdown module checking by considering environments with imperfect information about the system's state and pushdown store content. To this aim, we first have to define how a pushdown system keeps part of its internal configuration invisible to the environment and another part visible. In [PR79], a *private pushdown store automata* is defined to be a Turing machine with two tapes: a read only public (visible) one-way input tape, and a possibly private (invisible) work tape, simulating a pushdown store. Unfortunately, their definition is not suitable for our purpose as it allows for only two levels of information hiding: either the pushdown store and control state are completely visible, or completely invisible. The definition we use instead is an extension of the idea used for finite-state systems. Like in the finite case, we assume the control states are assignments to boolean *control variables*, some of which are visible and some of which are invisible. Similarly, symbols of the pushdown store are assignments to boolean visible and invisible *pushdown store variables*.

In [KV97], each state is partitioned into input, output, and invisible variables, where the environment supplies the input variables, and the system supplies the output and invisible variables. This idea works well for finite state-systems but not when we have to deal with imperfect information about the pushdown store. Note that a symbol pushed now, influences the computation much later, when it becomes the top of the pushdown store. Indeed, asking the environment to supply as input part of each symbol in the pushdown, is asking it to intimately participate in the internals of the computation, which is less natural. We find it more natural to think of the environment as choosing the possible transitions at certain points of the computation. For example, if the environment supplies the current reading of a physical sensor, we think of it as disabling all the transitions that are irrelevant for this reading. Thus, we model an open pushdown system with imperfect information by partitioning configurations into system and environment configurations, and also partitioning states and pushdown store symbols into visible and invisible variables, combining features from both [KV96] and [KV97].

We study the complexity of the pushdown module-checking problem with imperfect information, with respect to the branching-time logic *CTL*. We show that the problem is undecidable in the general case. We also show that undecidability relies on hiding information about the pushdown store. Indeed, we prove that *CTL* pushdown module checking with imperfect information about the internal control states, but a visible pushdown store, is decidable and 2EXPTIME-complete. Hence, it is not harder than perfect information *CTL* pushdown module checking. For the upper bound we use an automata-theoretic approach and introduce a new automata model, namely *semi-alternating pushdown Büchi tree automata* (*PD-SBT*). These are alternating pushdown Büchi tree automata [KPV02] where the universality is not allowed on the pushdown store content. That is, two copies of the automaton that read the same input, from two configurations that have the same top of pushdown store, must push the same value into the pushdown store. Our algorithm reduces the addressed problem to the emptiness problem of PD-SBT. We show that PD-SBT are equivalent to nondeterministic pushdown Büchi tree automata, for which the emptiness problem can be solved in EXPTIME [KPV02]. Note that alternating pushdown automata, in contrast to the semi-alternating ones, are *not* equivalent to nondeterministic pushdown automata. Indeed, since the emptiness problem of the intersection of two context free languages is undecidable [HU79], the emptiness problem of alternating pushdown automata is undecidable already in the case of finite words.

2 Preliminaries

Let Υ be a set. An Υ*-tree* is a prefix closed subset $T \subseteq \Upsilon^*$. The elements of T are called *nodes* and the empty word ε is the *root* of T. For $v \in T$, the set of *children* of v (in T) is $child(T, v) = \{v \cdot x \in T \mid x \in \Upsilon\}$. Given a node $v = u \cdot x$, with $u \in \Upsilon^*$ and $x \in \Upsilon$, we define $last(v)$ to be x. We also say that v *corresponds* to x. The complete Υ-tree is the tree Υ^*. For $v \in T$, a (full) path π of T from v is a *minimal* set $\pi \subseteq T$ such that $v \in \pi$ and for each $v' \in \pi$ such that $child(T, v') \neq \emptyset$, there is exactly one node in $child(T, v')$ belonging to π. Note that every infinite word $w \in \Upsilon^\omega$ can be thought of as an infinite path in the tree Υ^*, namely the path containing all the finite prefixes of w. For an alphabet Σ, a Σ-labeled Υ-tree is a pair $\langle T, V\rangle$ where T is an $\Upsilon-$tree and $V : T \to \Sigma$ maps each node of T to a symbol in Σ.

An *open system* is a system that interacts with its environment and whose behavior depends on this interaction. We consider the case where the environment has imperfect information about the system, i.e., when the system has internal variables that are not visible to its environment. We describe such a system by a *module* $\mathcal{M} = \langle AP, W_s, W_e, w_0, R, L, \cong\rangle$, where AP is a finite set of *atomic propositions*, W_s is a set of *system states*, and W_e is a set of *environment states.* We assume $W_s \cap W_e = \emptyset$, and call $W = W_s \cup W_e$ the set of $\mathcal{M}$'s *states.* $w_0 \in W$ is the *initial state*, $R \subseteq W \times W$ is a total *transition relation*, $L : W \to 2^{AP}$ is a labeling function that maps each state of $\mathcal{M}$ to the set of atomic propositions that hold in it, and $\cong$ is an equivalence relation on W.

In order to present a unified definition that is general enough to handle both finite-state and infinite-state systems, we model the fact that the environment has imperfect information about the states of the system by an equivalence relation $\cong$. States that are indistinguishable by the environment, because the difference between them is kept invisible by the system, are equivalent according to $\cong$. We write $[W]$ for the set of equivalence classes of W under $\cong$. Since states in the same equivalence class are indistinguishable by the environment, from the environment's point of view, the states of the system are actually the equivalence classes themselves. The equivalence class $[w]$ of $w \in W$, is called the *visible part* of w, since it is in a sense what the environment "sees" of w. We write $vis(w)$ instead of $[w]$, to emphasize this. Note that we can also do the converse. That is, given a function vis, whose domain is W, we can define the equivalence relation $\cong$ by letting $w \cong w'$ iff $vis(w) = vis(w')$. We can then think of the range of vis as the set of the equivalence classes $[W]$ and associate $[w]$ with the value $vis(w)$.

A module $\mathcal{M}$ is *closed* if $W_e = \emptyset$ (meaning that $\mathcal{M}$ does not interact with any environment) and *open* otherwise. Since the designation of a state as an environment state is obviously known to the environment, we require that for every $w, w' \in W$ such that $w \cong w'$, we have that $w \in W_e$ iff $w' \in W_e$. Also note that if $w \cong w'$, from the environment's point of view, the set of atomic propositions that currently hold in w may just as well be $L(w')$. We therefore define the labeling, as seen by the environment, as a function $visL : [W] \to 2^{2^{AP}}$ that maps the visible part of a state to a set of possible sets of atomic propositions: $visL([u]) = \{L(w) \mid w \in W \wedge w \cong u\}$. If it is always the case that $w \cong w' \implies L(w) = L(w')$, we say that the atomic propositions are visible.

For $\langle w, w' \rangle \in R$, we say that w' is a *successor* of w. The requirement that R be total means that every state w has at least one successor. A *computation* of $\mathcal{M}$ is a sequence $w_0 \cdot w_1 \cdots$ of states, such that for all $i \geq 0$ we have $\langle w_i, w_{i+1} \rangle \in R$. For each $w \in W$, we denote by $succ(w)$ the set (possibly empty) of w's successors. When the module $\mathcal{M}$ is in a system state w_s, then all successor states are possible next states. On the other hand, when $\mathcal{M}$ is in an environment state w_e, the environment decides, based on the visible parts of each successor of w_e, and of the history of the computation so far, to which of the successor states the computation can proceed, and to which it can not.

The set of all (maximal) computations of $\mathcal{M}$ starting from the initial state w_0 can be described by an AP-labeled W-tree $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$ called a *computation tree*, which is obtained by unwinding $\mathcal{M}$ in the usual way. Each node $v = v_1 \cdots v_k$ of $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$ describes the (partial) computation $w_0 \cdot v_1 \cdots v_k$ of $\mathcal{M}$, with the root ε corresponding to w_0. The children of v are exactly all nodes of the form $v_1 \cdots v_k \cdot w$, where w ranges over all the successors of v_k in $\mathcal{M}$. We extend the definition of the vis function to nodes in the natural way. Thus, the visible part of a node v is $vis(v) = vis(v_1) \cdots vis(v_k)$. The labeling $V_{\mathcal{M}}$ of a node v depends on the state it corresponds to (its last state), i.e., $V_{\mathcal{M}}(v) = L(last(v))$. Also, if v corresponds to an environment state, we say that v is an *environment node.*

The problem of deciding, for a given *CTL* formula[2] φ over the set AP of atomic propositions, whether $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$ satisfies φ is the usual *model checking problem* (formally denoted $\mathcal{M} \models \varphi$) [CE81, QS81]. In model checking, we only have to consider the computation tree $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$, since the module we want to check is closed and thus its behavior is not affected by the environment. On the other hand, whenever we consider an open module, $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$ corresponds to a very specific environment: a maximal environment that never restricts the set of next states. Therefore, when we examine a branching-time specification φ w.r.t. an open module $\mathcal{M}$, the formula φ should hold not only in $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$, but in all the trees obtained by pruning from $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$ subtrees whose roots are children (successors) of environment nodes, in accordance with all *possible* environments. It is important to note that in the case of perfect information (i.e., $\cong$ is actually the equality relation), every such pruning corresponds to some environment; however, in the case of imperfect information, only if the pruning is consistent with the partial information available to the environment, will the tree correspond to an actual environment. Formally, if two nodes v and v' are indistinguishable, i.e., if $vis(v) = vis(v')$, then a tree in which the subtree rooted at v is pruned, but the one rooted at v' is not pruned, does not correspond to any environment, and should not be considered. As noted in [KV97], the fact that given a pruning of $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$, a finite automaton cannot decide if that pruning corresponds to an actual environment or not, is the main source of difficulty in dealing with module checking with imperfect information. Also note that the knowledge-based subset construction that is used to transform games of imperfect information into ones of perfect information (see for example [CDHR06]), is not applicable in this context, since in general there is no connection between the satisfiability of a branching time formula on the original structure and its satisfiability on the one obtained by the knowledge-based subset construction.

Recall that whenever $\mathcal{M}$ interacts with an environment ξ, its possible moves from environment states depends on the behavior of ξ. We can think of an environment to $\mathcal{M}$ as a strategy $\xi : [W]^* \to \{\top, \bot\}$ that maps a finite history s of a computation, as seen by the environment, to either $\top$ or $\bot$, meaning that the environment respectively allows or disallows $\mathcal{M}$ to trace s. We say that the tree $\langle [W]^*, \xi \rangle$ maintains the strategy applied by ξ, and we call it a *strategy tree*. We denote by $\mathcal{M} \lhd \xi$ the AP-labeled W-tree induced by the composition of $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$ with ξ; that is, the AP-labeled W-tree obtained by pruning from $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$ subtrees according to ξ. Note that by the definition above, ξ may disable all the children of a node v. Since we usually do not want the environment to completely block the system, we require that at least one child of each node is enabled. In this case, we say that the composition $\mathcal{M} \lhd \xi$ is *deadlock free*.

To see the interaction of $\mathcal{M}$ with ξ, let $v \in T_{\mathcal{M}}$ be an environment node, and $v' \in T_{\mathcal{M}}$ be one of its children. The subtree rooted in v' is pruned iff $\xi(vis(v')) = \bot$. Every two nodes v_1 and v_2 that are indistinguishable according to ξ's imperfect information have $vis(v_1) = vis(v_2)$. Also, recall that the designation of a state as an environment state is based only on the visible part of

[2] For a definition of the syntax and semantics of *CTL* see for example [KV96].

that state. Thus, if v_1 is a child of an environment node then so is v_2, and either both subtrees with roots v_1 and v_2 are pruned, or both are not. Note that once $\xi(v) = \bot$ for some $v \in [W]^*$, we can ignore $\xi(v \cdot t)$, for all $t \in [W]^*$. Indeed, once the environment disables the transition to a certain node v, it actually disables the transitions to all the nodes in the subtree with root v. We can now formally define the interaction of an open module with an environment with imperfect information. From now on, unless stated differently, we always refer to modules that are open, and environments with imperfect information. Given a module $\mathcal{M}$, and a strategy tree $\langle [W]^*, \xi \rangle$ for an environment ξ, an AP-labeled W-tree $\langle T, V \rangle$ corresponds to $\mathcal{M} \lhd \xi$ iff the following hold:

– The root of T corresponds to w_0.
– For $v \in T$ with $last(v) \in W_s$, we have $child(T, v) = \{v \cdot w_1, \ldots, v \cdot w_n\}$, where $succ(last(v)) = \{w_1, \ldots, w_n\}$.
– For $v \in T$ with $last(v) \in W_e$, there exists a nonempty subset $\{w_1, \ldots, w_k\}$ of $succ(last(v))$ such that $child(T, v) = \{v \cdot w_1, \ldots, v \cdot w_k\}$. Furthermore, for all w in $\{w_1, \ldots, w_k\}$ we have that $\xi(vis(v \cdot w)) = \top$, while for all w in $succ(last(v)) \setminus \{w_1, \ldots, w_k\}$ we have that $\xi(vis(x \cdot w)) = \bot$.
– For every node $v \in T$, we have that $V(v) = L(last(v))$.

For a module $\mathcal{M}$ and a temporal logic formula over the set AP, we say that $\mathcal{M}$ *reactively satisfies* φ, denoted $\mathcal{M} \models_r \varphi$, if $\mathcal{M} \lhd \xi$ satisfy φ, for every environment ξ for which $\mathcal{M} \lhd \xi$ is deadlock free. The problem of deciding whether $\mathcal{M} \models_r \varphi$ is called *module checking*, and was first introduced and studied in [KV96, KVW01] for finite-state systems with perfect information. The problem was successively extended to imperfect information in [KV97]. For *CTL* formulas it has been shown that the complexity of both problems is EXPTIME-complete[3].

3 Imperfect Information Pushdown Module Checking

In this section, we extend the notion of module checking with imperfect information to infinite-state systems induced by *Open Pushdown Systems* (*OPD*).

Definition 1. *An* OPD *is a tuple* $\mathcal{S} = \langle AP, Q, q_0, \Gamma, \flat, \delta, \mu, Env \rangle$*, where* AP *is a finite set of atomic propositions,* Q *is the set of (*control*)* states*, and* $q_0 \in Q$ *is an* initial state*. We assume that* $Q \subseteq 2^{I \cup H}$ *where* I *and* H *are disjoint finite sets of* visible *and* invisible control variables*, respectively.* Γ *is a finite pushdown store alphabet,* $\flat \notin \Gamma$ *is the* pushdown store bottom symbol*, and we use* $\Gamma_\flat$ *to denote* $\Gamma \cup \{\flat\}$*. We assume that* $\Gamma \subseteq 2^{I_\Gamma \cup H_\Gamma}$ *where* I_Γ *and* H_Γ *are disjoint finite sets of* visible *and* invisible pushdown store variables*, respectively.* $\delta \subseteq (Q \times \Gamma_\flat) \times (Q \times \Gamma_\flat^*)$ *is a finite transition relation, and* $\mu : Q \times \Gamma_\flat \to 2^{AP}$ *is a labeling function.* $Env \subseteq Q \times \Gamma_\flat$ *is used to specify the set of* environment configurations*. The size* $|\mathcal{S}|$ *of* $\mathcal{S}$ *is* $|Q| + |\Gamma| + |\delta|$*, with* $|\delta| = \sum_{((p,\gamma),(q,\beta)) \in \delta} |\beta|$.

[3] Although the complexity of the perfect and imperfect information cases coincide in the general case, [KVW01, KV97] show that when the formula is constant the imperfect information case is exponentially harder.

A *configuration* of $\mathcal{S}$ is a pair (q, α) where q is a control state and $\alpha \in \Gamma^* \cdot \flat$ is a pushdown store content. We write $top(\alpha)$ for the leftmost symbol of α and call it the *top of the pushdown store* α. The *OPD* moves according to the transition relation. Thus, $((p, \gamma), (q, \beta)) \in \delta$ implies that if the *OPD* is in state p and the top of the pushdown store is γ, it can move to state q, pop γ and push β. We assume that if $\flat$ is popped it gets pushed right back, and that it only gets pushed in such cases. Thus, $\flat$ is always present at the bottom of the pushdown store, and nowhere else. Note that we make this assumption also about the various pushdown automata we use later. Also note that the possible moves of the system, the labeling function, and the designation of configurations as environment configurations, are all dependent only on the current control state and the top of the pushdown store.

For a control state $q \in Q$, the visible part of q is $vis(q) = q \cap I$. For a pushdown store symbol $\gamma \in \Gamma$, if $\gamma \subseteq H_\Gamma$ and $\gamma \neq \emptyset$ we set $vis(\gamma) = \varepsilon$, otherwise we set $vis(\gamma) = \gamma \cap I_\Gamma$. By setting $vis(\gamma) = \varepsilon$ whenever γ consists entirely of invisible variables, we allow the system to completely hide a push operation (obviously a corresponding pop will also be invisible). When such a push occurs, the environment does not see the symbol $\emptyset$ being pushed, rather, it sees no push at all. This is necessary since in many applications what is actually pushed is immaterial, and the information to be revealed or hidden is only the depth of the pushdown store. The visible part of a pushdown store content $s = \gamma_0 \cdots \gamma_n \cdot \flat$ is defined in the natural way: $vis(s) = vis(\gamma_0) \cdots vis(\gamma_n) \cdot \flat$. The visible part of a configuration (q, α), is thus $vis((q, \alpha)) = (vis(q), vis(\alpha))$. As for modules, the designation of a configuration of an *OPD* as an environment configuration is known to the environment. Thus, we require that for every two configurations (q, α) and (q', α') such that $vis(q, top(\alpha)) = vis(q', top(\alpha'))$, it holds that $(q, top(\alpha)) \in Env$ iff $(q', top(\alpha')) \in Env$.

Definition 2. *An* OPD $\mathcal{S} = \langle AP, Q, q_0, \Gamma, \flat, \delta, \mu, Env \rangle$ *induces an infinite-state module* $\mathcal{M}_S = \langle AP, W_s, W_e, w_0, R, L, \cong \rangle$*, where:*

- *AP is a set of atomic propositions;*
- *$W_s \cup W_e = Q \times \Gamma^* \cdot \flat$ is the set of configurations;*
- *W_e is the set of configurations (q, α) such that $(q, top(\alpha)) \in Env$;*
- *$w_0 = (q_0, \flat)$ is the initial configuration;*
- *R is a transition relation, where $((q, \gamma \cdot \alpha), (q', \beta)) \in R$* iff *there exist $((q, \gamma), (q', \beta')) \in \delta$ such that $\beta = \beta' \cdot \alpha$;*
- *$L((q, \alpha)) = \mu(q, top(\alpha))$ for all $(q, \alpha) \in W$;*
- *For every $w, w' \in W$, we have that $w \cong w'$ iff $vis(w) = vis(w')$.*

To describe the interaction of an *OPD* $\mathcal{S}$ with its environment, we consider the interaction of the environment with the induced module $\mathcal{M}_S$. Indeed, every environment ξ of $\mathcal{S}$, can be represented by a strategy tree $\langle [W]^*, \xi \rangle$, and the composition $\mathcal{M}_S \lhd \xi$ of $\langle [W]^*, \xi \rangle$ with $\langle T_{\mathcal{M}_S}, V_{\mathcal{M}_S} \rangle$ describes all the computations of $\mathcal{S}$ allowed by the environment ξ. We can thus define the following problem.

Pushdown module checking problem with imperfect information: Given an *OPD* $\mathcal{S}$ and a *CTL* formula[4] φ, decide whether $\mathcal{M}_S \models_r \varphi$, i.e., whether $\mathcal{M}_S \lhd \xi$ satisfy φ, for every environment ξ for which $\mathcal{M}_S \lhd \xi$ is deadlock free.

Note that starting with an *OPD* $\mathcal{S}$ having $Env = \emptyset$ (that is, the behavior of $\mathcal{S}$ is not affected by any environment) the induced module is closed. In this case, the problem we address becomes the classical *pushdown model checking problem*, and for *CTL* specifications it has been studied in [Wal96, Wal00]. Also, if the *OPD* is open ($Env \neq \emptyset$) but there is no invisible information (both H and H_Γ are empty), the addressed problem is called *pushdown module checking with perfect information*, and it has been studied in [BMP05].

In the remaining part of this section, we study the pushdown module checking problem with imperfect information and show that it is undecidable for *CTL* specifications. In the next section, we show that undecidability relies on the system's ability to hide information about the pushdown store. Namely, we prove that if we start with an *OPD* with $H_\Gamma = \emptyset$, the problem becomes decidable (even if $H \neq \emptyset$), and its complexity is the same as that of pushdown module checking with perfect information.

Undecidability of the pushdown module checking problem with imperfect information is obtained by a reduction from the universality problem of nondeterministic pushdown automata on finite words (*PDA*), which is undecidable [HU79]. That is, given a *PDA* $\mathcal{P}$, we build an *OPD* $\mathcal{S}$ and a *CTL* formula φ, such that the module induced by $\mathcal{S}$ reactively satisfies φ iff $\mathcal{P}$ is universal.

Our choice to do a reduction from the universality problem of PDA is not at all arbitrary[5]. It is well known that checking for the universality of a nondeterministic automaton can be thought of as a game between a protagonist trying to prove that the automaton is not universal, and an antagonist claiming that it is universal. The universality game is played as follows. The protagonist chooses the first symbol, the antagonist responds with the first part of the run, the protagonist chooses the next symbol, the antagonist extends the run, and so on. The protagonist wins if the resulting run is rejecting, and the antagonist wins if it is accepting. Note that if the automaton is not universal the protagonist has a winning strategy, namely, choosing the letters of a word not accepted by the automaton. However, since the automaton is nondeterministic, the converse is not true. That is, even if the automaton is universal, the antagonist may not have a winning strategy. Also note that (again due to nondeterminism) if the protagonist can see the moves of the antagonist, it may force the run to be rejecting even though the word it supplies can be accepted by the automaton. Hence, the game is sound but not complete. However, if the protagonist cannot see the moves of the antagonist the game becomes sound and complete. Deciding if the

[4] The semantics of *CTL* is usually defined with respect to infinite paths, so we assume $\mathcal{M}_S$ has no configurations without successors. However, using a similar technique to the one used in [BMP05] our results can be adapted to the situation where terminal configurations are also allowed.

[5] We thank Martin Lange for a useful discussion on the connection between the proof of Theorem 1 and the game interpretation of the universality problem.

automaton is not universal can be reduced to deciding whether the antagonist has a winning strategy in the corresponding universality game with imperfect information. By casting the universality game of *PDA* to a special instance of the pushdown module checking problem with imperfect information, the latter is shown to be undecidable. The complete proof can be found in the full version.

Theorem 1. *The pushdown module-checking problem with imperfect information for* CTL *specifications is undecidable.*

It turns out that even if the environment has full information about the control states and (surprisingly enough) about which atomic propositions hold at each configuration the problem remains undecidable. Thus, we have.

Theorem 2. *The imperfect information pushdown module checking problem for* CTL*, with visible control states and atomic propositions, is undecidable.*

4 Module Checking with Visible Pushdown Store

In this section, we show that pushdown module checking for *CTL* with full information about the pushdown store content ($H_{\Gamma} = \emptyset$), but not about the control states (when $H \neq \emptyset$), is decidable and 2EXPTIME-complete, matching the complexity of pushdown module checking with complete information. For the upper bound we use an automata-theoretic approach and introduce a new automata model, namely *semi-alternating pushdown Büchi tree automata* (*PD-SBT*). Our algorithm reduces the addressed problem to the emptiness problem of PD-SBT. We show that PD-SBT are equivalent to nondeterministic pushdown Büchi tree automata, for which emptiness can be decided in EXPTIME[KPV02]. The formal definition of semi-alternating pushdown tree automata follows.

Semi-alternating Pushdown Tree Automata. A *PD-SBT* is a tuple $\mathcal{A} = \langle \Sigma, D, \Gamma, Q, q_0, \flat, \delta, F \rangle$ where Σ is a finite input alphabet, D is a finite set of *directions*, Γ is a finite pushdown store alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $\flat \notin \Gamma$ is the pushdown store bottom symbol, and $F \subseteq Q$ is a Büchi acceptance condition. Moreover, δ is a finite transition relation defined as a function $\delta : Q \times \Sigma \times \Gamma_{\flat} \rightarrow \mathcal{B}^{+}(D \times Q \times \Gamma_{\flat}^{*})$, where $\Gamma_{\flat} = \Gamma \cup \{\flat\}$ as usual, and $\mathcal{B}^{+}(D \times Q \times \Gamma_{\flat}^{*})$ is the set of all finite positive boolean combinations of triples (d, q, β), where d is a direction, q is a state, and β is a string of pushdown store symbols. We also allow the formulas **true** and **false**. We write $S \in \delta(p, \sigma, \gamma)$ to denote that S is a set of tuples (d, q, β) that satisfy $\delta(p, \sigma, \gamma)$.

What makes the automaton semi-alternating is the requirement that for every $d \in D$, $\sigma \in \Sigma$, $p, p' \in Q$ (possibly the same state), and $\gamma \in \Gamma$, if (d, q, β) appears in $\delta(p, \sigma, \gamma)$, and (d, q', β') appears in $\delta(p', \sigma, \gamma)$, then $\beta = \beta'$. That is, two copies of the automaton that read the same input, from two configurations that have the same top symbol of the pushdown store and proceed in the same direction, must push the same value into the pushdown store. In particular, it follows that in every run, two copies of the automaton that are reading the same node of an input tree have the same pushdown store content. Note that if we remove the semi-alternation requirement, the resulting automaton is called *alternating pushdown Büchi tree automaton* (*PD-ABT*).

As an example, for $D = \{0, 1\}$, having $\delta(q, \sigma, \gamma) = ((0, q_1, \beta_1) \vee (1, q_2, \beta_2)) \wedge (1, q_1, \beta_2)$ means that when a copy of the automaton that is in a configuration where the current state is q, and the top of pushdown store is γ, reads a node in the input tree whose label is σ, it can proceed in one of two ways. In the first case, one copy proceeds in direction 0 to state q_1, by replacing γ with β_1, and one copy proceeds in direction 1 to state q_1, by replacing γ with β_2. In the second case, two copies proceed in direction 1, one to state q_1 and the other to state q_2, and in both copies γ is replaced with β_2. Hence, $\vee$ and $\wedge$ in $\delta(q, \sigma, \gamma)$ represent, respectively, choice and concurrency. As a special case of PD-ABT, we consider *nondeterministic pushdown Büchi tree automata* (*PD-NBT*) where the concurrency feature is not allowed. That is, whenever a PD-NBT visits a node x of the input tree, it sends to each successor (direction) of x at most one copy of itself. More formally, a PD-NBT is a PD-ABT in which δ is in disjunctive normal form, and in each conjunct each direction appears at most once.

A run of a PD-SBT $\mathcal{A}$ on a Σ-labeled tree $\langle T, V\rangle$, with $T = D^*$, is a $(D^* \times Q \times \Gamma^* \cdot \flat)$-labeled $\mathbb{N}$-tree $\langle T_r, r\rangle$ such that the root is labeled with $(\varepsilon, q_0, \flat)$ and the labels of each node and its successors satisfy the transition relation. Formally, a $(D^* \times Q \times \Gamma^* \cdot \flat)$-labeled tree $\langle T_r, r\rangle$ is a run of $\mathcal{A}$ on $\langle T, V\rangle$ iff

- $r(\varepsilon) = (\varepsilon, q_0, \flat)$, and
- for all $x \in T_r$ such that $r(x) = (y, p, \gamma \cdot \alpha)$, there is an $n \in \mathbb{N}$ such that the successors of x are exactly $x\cdot 1, \ldots x\cdot n$, and for all $1 \le i \le n$ we have $r(x\cdot i) = (y \cdot d_i, p_i, \beta_i \cdot \alpha)$ for some $\{(d_1, p_1, \beta_1), \ldots, (d_n, p_n, \beta_n)\} \in \delta(p, V(y), \gamma)$.

For a path $\pi \subseteq T_r$, let $inf_r(\pi) \subseteq Q$ be the set of states that appear in the labels of infinitely many nodes in π. For a Büchi condition $F \subseteq Q$, we have that π is *accepting* iff $inf_r(\pi) \cap F \neq \emptyset$. A run $\langle T_r, r\rangle$ is *accepting* iff all its paths are accepting. The automaton $\mathcal{A}$ accepts an input tree $\langle T, V\rangle$ iff there is an accepting run of $\mathcal{A}$ over $\langle T, V\rangle$. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of Σ-labeled trees accepted by $\mathcal{A}$. We say that an automaton $\mathcal{A}$ is nonempty iff $L(\mathcal{A}) \neq \emptyset$.

Given a PD-SBT $\mathcal{A} = \langle \Sigma, D, \Gamma, Q, q_0, \flat, \delta, F\rangle$, we define the size of $\mathcal{A}$ as $|\mathcal{A}| = |Q| + |\delta|$, where $|\delta|$ is the sum of the lengths of the satisfiable (i.e., not **false**) formulas that appear in $\delta(q, \sigma, \gamma)$ for some q, σ, and γ.

In [MH84], Miyano and Hayashi describe a translation of alternating Büchi automata on words to nondeterministic ones. We now present an extension of their translation to show the equivalence of PD-SBT and PD-NBT.

Lemma 1. *Let $\mathcal{A}$ be a PD-SBT with n states. There is a PD-NBT $\mathcal{A}'$ with $2^{O(n)}$ states, such that $L(\mathcal{A}') = L(\mathcal{A})$.*

Proof. The automaton $\mathcal{A}'$ guesses a subset construction applied to a run of $\mathcal{A}$. At a given node x of a run of $\mathcal{A}'$, it keeps in its memory the set of configurations in which the various copies of $\mathcal{A}$ visit node x in the guessed run. Since $\mathcal{A}$ is semi-alternating, all copies of $\mathcal{A}$ that visit the same node x have the same pushdown store content, and thus can all be remembered using one pushdown store and a set of states of $\mathcal{A}$. In order to make sure that every infinite path visits states in F infinitely often, $\mathcal{A}'$ keeps track of states that "owe" a visit to F. Let $\mathcal{A} = \langle \Sigma, D, \Gamma, Q, q_0, \flat, \delta, F\rangle$. Then $\mathcal{A}' = \langle \Sigma, D, \Gamma, 2^Q \times 2^Q, \langle \{q_0\}, \emptyset\rangle, \flat, \delta', 2^Q \times \{\emptyset\}\rangle$, where δ' is defined as follows. We first need the following notation. For a set $S \subseteq Q$, a letter $\sigma \in \Sigma$, and a top of pushdown store symbol $\gamma \in \Gamma$, let $sat(S, \sigma, \gamma)$ be the set of subsets of $D \times Q \times \Gamma_\flat^*$ that satisfy $\bigwedge_{q \in S} \delta(q, \sigma, \gamma)$. Also, for two sets $O \subseteq S \subseteq Q$, a letter $\sigma \in \Sigma$, and a top of pushdown store symbol $\gamma \in \Gamma$, let $pair\_sat(S, O, \sigma, \gamma)$ be such that $\langle S', O'\rangle \in pair\_sat(S, O, \sigma, \gamma)$ iff $S' \in sat(S, \sigma, \gamma)$, $O' \subseteq S'$, and $O' \in sat(O, \sigma, \gamma)$. Finally, for a direction $d \in D$, we have $S'_d = \{s \mid (d, s, \beta) \in S'$ for some $\beta\}$ and $O'_d = \{o \mid (d, o, \beta) \in O'$ for some $\beta\}$. Thus, S'_d and O'_d are, respectively, the collections of all states that appear in S' and O' along with the direction d. Since $\mathcal{A}$ is semi-alternating, for every two triplets (d, q, β) and (d, q', β') in $sat(S, \sigma, \gamma)$ having the same direction d, we have that $\beta = \beta'$. Thus, we can define $store(d, \sigma, \gamma) = \beta$.

Now, δ' is defined, for all $\langle S, O\rangle \in 2^Q \times 2^Q$, $\sigma \in \Sigma$, and $\gamma \in \Gamma$, as follows.

– if $O \neq \emptyset$, then

$$\delta'(\langle S, O\rangle, \sigma, \gamma) = \bigvee_{\substack{\langle S', O'\rangle \in \\ pair\_sat(S, O, \sigma, \gamma)}} \bigwedge_{d \in D} (d, \langle S'_d, O'_d \setminus F\rangle, store(d, \sigma, \gamma))$$

Thus, when reading σ, from a configuration with a top of pushdown store symbol γ, the automaton $\mathcal{A}'$ sends to a direction $d \in D$ the set S'_d of states that the different copies of $\mathcal{A}$ send to direction d in the guessed run. Each such S'_d is paired with a subset O'_d of S'_d of the states that still "owe" a visit to F. The key observation is that since $\mathcal{A}$ is semi-alternating, all the copies that $\mathcal{A}$ sends to direction d replace γ with exactly the same pushdown store string, namely, with $store(d, \sigma, \gamma)$. Hence, the pushdown stores of all the copies that $\mathcal{A}$ sends to direction d are identical, and $\mathcal{A}'$ can keep track of them all using the single stack of the copy it send to direction d.

– if $O = \emptyset$, then

$$\delta'(\langle S, O\rangle, \sigma, \gamma) = \bigvee_{\substack{\langle S', O'\rangle \in \\ pair\_sat(S, O, \sigma, \gamma)}} \bigwedge_{d \in D} (d, \langle S'_d, S'_d \setminus F\rangle, store(d, \sigma, \gamma))$$

Thus, when no state "owes" a visit to F we know that every path in the guessed run of $\mathcal{A}$ visited F one more time, and the requirement to visit F is reinforced. □

We can now show decidability for pushdown module checking for *CTL* with visible pushdown store. The decidability follows from Lemma 1, the fact that emptiness of PD-NBT is decidable, and the following theorem.

Theorem 3. *For an OPD $\mathcal{S}$ with $H_\Gamma = \emptyset$ and a CTL formula φ over $\mathcal{S}$'s atomic propositions, there is a PD-SBT $\mathcal{A}_{\mathcal{S},\varphi}$ of size $O(|\mathcal{S}| * |\varphi|)$, such that $L(A_{\mathcal{S},\varphi})$ is the set of strategies ξ such that $\mathcal{M}_S \lhd \xi$ is deadlock free and satisfies φ.*

Proof (Sketch). Essentially, the automaton $\mathcal{A}_{\mathcal{S},\varphi}$ we build is an extension of the product automaton obtained in the alternating-automata theoretic approach for *CTL* module checking with imperfect information [KV97]. The extension we consider here concerns the simulation of the pushdown store of the *OPD*.

Let $\mathcal{S} = \langle AP, Q, q_0, \Gamma, \flat, \delta, \mu, Env\rangle$ be an OPD, let φ be a *CTL* formula in positive normal form, and let $\mathcal{M}_S = \langle AP, W_s, W_e, w_0, R, L, \cong\rangle$ be the module induced by $\mathcal{S}$. We build an automaton $\mathcal{A}_{\mathcal{S},\varphi}$ that accepts $\{\top, \bot\}$-labeled trees corresponding to strategies ξ, whose composition with $\mathcal{M}_S$ is deadlock free and satisfy φ. Intuitively, a run of $\mathcal{A}_{\mathcal{S},\varphi}$ on an input strategy tree ξ, proceeds by simulating an unwinding of the module $\mathcal{M}_S$, pruned at each step according to the strategy ξ. Copies of the automaton simulating nodes in the computation tree of $\mathcal{M}_S$ that are indistinguishable by the environment are sent to the same direction in the input tree. The resulting run tree of $\mathcal{A}_{\mathcal{S},\varphi}$ on ξ is basically a replica of the composition $\mathcal{M}_S \lhd \xi$, and the fact that it satisfies the formula φ is checked on the fly, by employing in $\mathcal{A}_{\mathcal{S},\varphi}$ the usual alternating-automata approach for *CTL* model checking. In the full computation tree of $\mathcal{M}_S$, the set of directions is $G = \{(q, \beta) \mid ((p, \alpha), (q, \beta)) \in R$ for some p, α and $\beta\}$. Since in $\mathcal{S}$ the pushdown store is completely visible to the environment, the set of directions of the input strategy trees is $D = \{(vis(q), \beta) \mid ((p, \alpha), (q, \beta)) \in R$ for some p, α and $\beta\}$. Finally, due to the fact that all copies of the automaton sent to direction $(vis(q), \beta)$ push β into the pushdown store, the resulting automaton $\mathcal{A}_{\mathcal{S},\varphi}$ is semi-alternating.

We formally define $\mathcal{A}_{\mathcal{S},\varphi} = \langle\{\top, \bot\}, D, \Gamma, Q', q'_0, \flat, \delta', F\rangle$, where

– $Q' = (Q \times (cl(\varphi) \cup \{p_\top\}) \times \{\forall, \exists\} \times \{p_e, p_s\}) \cup \{q'_0\}$. States with the component $p_\top$ are used to check that the composition of $\mathcal{M}_S$ with the strategy is deadlock free, while states with a component in $cl(\varphi)$ check that this composition satisfies φ. The components p_e and p_s are used to flag that a currently simulated node, of the computation tree of $\mathcal{M}_S$, is a child of an environment or a system node, respectively. Clearly, the simulation should respect the strategy pruning specifications only if they correspond to children of environment nodes; that is, only if the current state q contains p_e. Every state is either in an existential or a universal mode, as specified by the $\forall$ and $\exists$ components. When the automaton is in a universal state $(q, \varphi, \forall, p_e)$ with a pushdown store content α, it accepts all strategies for which (q, α) in $\mathcal{M}_S$ is either pruned or satisfies φ (where $p_\top$ is satisfied iff the root of the strategy is labeled $\top$). When the automaton is in an existential state $(q, \varphi, \exists, p_e)$ with a pushdown store content α, it accepts all strategies for which (q, α) in $\mathcal{M}_S$ is not pruned and satisfies φ.

– The formal definition of $\delta' : Q' \times \Sigma \times \Gamma_\flat \rightarrow \mathcal{B}^+(D \times Q' \times \Gamma_\flat^*)$ is reported in the full version. Here, we just give an example of a transition rule. Consider, a transition from the configuration $(\langle p, \forall X\psi, \exists, p_e\rangle, \gamma \cdot \alpha)$, where $(p, \gamma) \in Env$. First, if the transition to $(p, \gamma\cdot\alpha)$ is disabled (that is, the automaton reads $\bot$), then, as the current mode is existential, the run is rejecting. If the transition to $(p, \gamma\cdot\alpha)$ is enabled, then the successors of $(p, \gamma\cdot\alpha)$ that are enabled should satisfy ψ. Note that all the successors of $(p, \gamma \cdot \alpha)$ that are indistinguishable by the environment are sent by the automaton to the same direction v. This

guarantees that either all these successors are enabled by the strategy (in case the letter to be read in direction v is $\top$) or all are disabled (in case the letter in direction v is $\bot$). In addition, since the requirement to satisfy ψ concerns only successors of $(p, \gamma \cdot \alpha)$ that are enabled, the mode of the new states is universal. The copies of $\mathcal{A}_{\mathcal{S},\varphi}$ that check the composition with the strategy to be deadlock free guarantee that at least one successor of $(p, \gamma \cdot \alpha)$ is enabled. As noted earlier, the enable/disable instructions of the strategy are ignored in every configuration $(p, \gamma \cdot \alpha)$ that is a successor of a system configuration. Also note that since we assume that no configuration in $\mathcal{M}_S$ has no successors, the conjunctions and disjunctions in δ' cannot be empty.
– $F = Q \times \widetilde{U}(\varphi) \times \{\exists, \forall\} \times \{p_e, p_s\}$, where $\widetilde{U}(\varphi)$ is the set of all formulas of the form $\forall \psi_1 \widetilde{U} \psi_2$ or $\exists \psi_1 \widetilde{U} \psi_2$ in $cl(\varphi)$.

In the full version we prove that $\mathcal{A}_{\mathcal{S},\varphi}$ is semi-alternating and that the size of δ' is $O(|\delta| * |\varphi|)$. Since $|Q'| = O(|Q| * |\varphi|)$, the size of $\mathcal{A}_{\mathcal{S},\varphi}$ is $O(|\mathcal{S}| * |\varphi|)$. □

We now consider the complexity bounds that follow from our algorithm.

Theorem 4. CTL *pushdown module checking with imperfect information about the control states but a visible pushdown store is* 2EXPTIME-*complete.*

Proof. The lower bound follows from the known bound for *CTL* pushdown module checking with perfect information [BMP05]. For the upper bound, Theorem 3 implies that $\mathcal{M}_S \models_r \varphi$ iff the language of the automaton $\mathcal{A}_{\mathcal{S},\neg\varphi}$ is empty. We recall that $\mathcal{A}_{\mathcal{S},\neg\varphi}$ is a PD-SBT of size $O(|\mathcal{S}| * |\varphi|)$. By Lemma 1, we can obtain a PD-NBT $\mathcal{A}$ equivalent to $\mathcal{A}_{\mathcal{S},\varphi}$, with an exponential blow-up. By [KPV02], the emptiness of $\mathcal{A}$ can be checked in exponential time. Thus, checking the emptiness of $\mathcal{A}$ is double-exponential in the sizes of $|\mathcal{S}|$ and $|\varphi|$. □

5 Discussion

We have shown that the pushdown module checking problem with imperfect information is undecidable for specifications given in *CTL*. Moreover, since the formula used in the proof of Theorems 1 and 2 is an existential formula, the problem is already undecidable for the existential fragment *ECTL* of *CTL*. This obviously implies the undecidability of the problem with respect to more expressive logics such as *CTL** and μ-calculus. Recall that in our setting, whenever we push a symbol consisting entirely of invisible variables, the environment does not see the push at all. One can think of a variant of the problem where the environment does see that a push occurred, but not what was pushed. Thus, the depth of the stack is always known to the environment. It is an open question whether this variant of the problem is decidable or not. As good news, we also showed that if the pushdown store is visible, the problem is decidable, and not harder than perfect information pushdown module checking. An interesting question is whether this variant of the problem remains decidable also for more expressive logics like *CTL**. By using an approach similar to the one used for *CTL*, we can reduce the problem for *CTL** to the emptiness problem of a semi-alternating pushdown tree automaton, but with a stronger acceptance condition, such as the parity condition. We do not know, however, if the emptiness problem for such automata is decidable or not. The main source of difficulty is that all known methods to remove alternation from parity finite tree automata involve a co-determinization step, and thus can not be easily adapted to pushdown automata. Even in [KV05] where the emptiness problem of alternating parity tree automata is reduced to that of nondeterministic automata, without a co-determinization step, the correctness *proof* of the construction does contain such a step. Nevertheless, it is our conjecture that despite these difficulties, *CTL** pushdown module checking with visible pushdown store is decidable.

References

[ABE$^+$05] Alur, R., Benedikt, M., Etessami, K., Godefroid, P., Reps, T.W., Yannakakis, M.: Analysis of recursive state machines. ACM Trans. Program. Lang. Syst. 27(4), 786–818 (2005)

[BEM97] Bouajjani, A., Esparza, J., Maler, O.: Reachability Analysis of Pushdown Automata: Application to Model-Checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997)

[BMP05] Bozzelli, L., Murano, A., Peron, A.: Pushdown module checking. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 504–518. Springer, Heidelberg (2005)

[CDHR06] Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.: Algorithms for omega-regular games with imperfect information. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 287–302. Springer, Heidelberg (2006)

[CE81] Clarke, E.M., Emerson, E.A.: Design and verification of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logics of Programs. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)

[CH05] Chatterjee, K., Henzinger, T.A.: Semiperfect-information games. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 1–18. Springer, Heidelberg (2005)

[EKS03] Esparza, J., Kucera, A., Schwoon, S.: Model checking LTL with regular valuations for pushdown systems. Inf. Comput. 186(2), 355–376 (2003)

[Hoa85] Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs (1985)

[HP85] Harel, D., Pnueli, A.: On the development of reactive systems. In: Logics and Models of Concurrent Systems. NATO Advanced Summer Institutes, vol. F-13, pp. 477–498. Springer, Heidelberg (1985)

[HU79] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)

[KPV02] Kupferman, O., Piterman, N., Vardi, M.Y.: Pushdown specifications. In: Baaz, M., Voronkov, A. (eds.) LPAR 2002. LNCS (LNAI), vol. 2514, pp. 262–277. Springer, Heidelberg (2002)

[KV96] Kupferman, O., Vardi, M.Y.: Module checking. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 75–86. Springer, Heidelberg (1996)

[KV97] Kupferman, O., Vardi, M.Y.: Module checking revisited. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 36–47. Springer, Heidelberg (1996)

[KV05] Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: IEEE FOCS'05, Pittsburgh, pp. 531–540. IEEE Computer Society Press, Los Alamitos (2005)

[KVW01] Kupferman, O., Vardi, M.Y., Wolper, P.: Module Checking. Information and Computation 164(2), 322–344 (2001)

[PR79] Peterson, G.L., Reif, J.H.: Multiple-person alternation. In: FOCS'79, pp. 348–363. IEEE Computer Society Press, Los Alamitos (1979)

[QS81] Queille, J.P., Sifakis, J.: Specification and verification of concurrent programs in Cesar. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) International Symposium on Programming. LNCS, vol. 137, pp. 337–351. Springer, Heidelberg (1982)

[Wal96] 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)

[Wal00] Walukiewicz, I.: Model checking CTL properties of pushdown systems. In: Kapoor, S., Prasad, S. (eds.) FST TCS 2000: Foundations of Software Technology and Theoretical Science. LNCS, vol. 1974, pp. 127–138. Springer, Heidelberg (2000)

Alternating Automata and a Temporal Fixpoint Calculus for Visibly Pushdown Languages

Laura Bozzelli

Università di Napoli Federico II , Via Cintia, 80126 - Napoli, Italy

Abstract. We investigate various classes of alternating automata for visibly pushdown languages (*VPL*) over infinite words. First, we show that alternating visibly pushdown automata (*AVPA*) are exactly as expressive as their nondeterministic counterpart (*NVPA*) but basic decision problems for *AVPA* are 2EXPTIME-complete. Due to this high complexity, we introduce a new class of alternating automata called *alternating jump automata* (*AJA*). *AJA* extend classical alternating *finite-state* automata over infinite words by also allowing *non-local* moves. A non-local forward move leads a copy of the automaton from a call input position to the matching-return position. We also allow local and non-local backward moves. We show that one-way *AJA* and two-way *AJA* have the same expressiveness and capture exactly the class of *VPL*. Moreover, boolean operations for *AJA* are easy and basic decision problems such as emptiness, universality, and pushdown model-checking for parity two-way *AJA* are EXPTIME-complete. Finally, we consider a linear-time fixpoint calculus which subsumes the full linear-time μ-calculus (with both forward and backward modalities) and the logic CARET and captures exactly the class of *VPL*. We show that formulas of this logic can be *linearly* translated into parity two-way *AJA*, and vice versa. As a consequence satisfiability and pushdown model checking for this logic are EXPTIME-complete.

1 Introduction

An active field of research is model-checking of pushdown systems. These represent an infinite-state formalism suitable to model the control flow of recursive sequential programs. The model checking problem of pushdown systems against regular properties is decidable and it has been intensively studied in recent years leading to efficient verification algorithms and tools (see for example [16,7,6,9]).

For context-free properties, the pushdown model checking problem is in general undecidable. However, algorithmic solutions have been proposed for checking interesting classes of context-free properties [9,10,8,3,4,1]. In particular, the linear temporal logic CARET, a context-free extension of *LTL*, has been recently introduced [3] which preserves decidability of pushdown model checking. CARET formulas are interpreted on infinite words over an alphabet (called *pushdown alphabet*) which is partitioned into three disjoint sets of calls, returns, and internal symbols. A call denotes invocation of a procedure (i.e. a push stack-operation) and the *matching* return (if any) along a given word denotes the exit from this procedure (corresponding to a pop stack-operation). CARET extends *LTL* by also allowing non-regular versions of the standard *LTL* temporal modalities: the *abstract modalities* can specify non-regular context-free properties which require matching of calls and returns such as correctness of procedures with respect to pre and post conditions, while the (backward) *caller modalities* are useful to express a variety of security properties that require inspection of the call-stack [9,10,8]. In [4], the class of *nondeterministic visibly pushdown automata* (*NVPA*) is proposed as an automata theoretic generalization of CARET. *NVPA* are pushdown automata which push onto the stack only when a call is read, pops the stack only at returns, and do not use the stack on reading internal symbols. Hence, the input controls the kind of stack operations which can be performed. The resulting class of languages (*visibly pushdown languages* or *VPL*, for short) includes strictly the class of regular languages and that defined by CARET and is robust like the class of regular languages. In particular, *VPL* are closed under all boolean operations and problems such as universality and inclusion that are undecidable for context-free languages are EXPTIME-complete for *VPL*.

Our contribution. We further investigate the class of *VPL*. We study various classes of alternating automata for *VPL* and derive interesting connections between a special class of two-way alternating finite-state automata and a fixpoint calculus for *VPL* with both forward and backward modalities. Note that the introduction of backward modalities in fixpoint logics can pose some difficulty in developing decision procedures for such logics since the interaction of backward modalities with the other modalities can be quite subtle. For example, while the standard modal μ-calculus has the finite-model property, this does not hold for the modal μ-calculus extended with backward modalities [15].

Alternating automata [12], i.e. automata featuring nondeterministic as well as universal choices, are interesting for many aspects[1]. For example, boolean operations, in particular complementation, are easy [12]. Moreover, alternating *finite-state* automata over words or trees have been founded to be particularly useful to derive optimal decision procedures for various regular temporal logics.

In this paper, first, we consider the alternating version of visibly pushdown automata. While unrestricted alternating pushdown automata on infinite words are more expressive than their nondeterministic counterpart (in particular, emptiness is undecidable), *alternating visibly pushdown automata* (*AVPA*) are exactly as expressive as *NVPA*: any parity *AVPA* $\mathcal{P}$ can be translated into an equivalent Büchi *NVPA* whose size is *doubly* exponential in the size of $\mathcal{P}$. This double-exponential blowup cannot be avoided. In fact we show that emptiness for parity or Büchi *AVPA* is 2EXPTIME-complete (recall that emptiness for parity *NVPA* is in PTIME [4]). As a consequence the pushdown model checking problem against *AVPA*-specifications is 2EXPTIME-complete.

Due to the high complexity of basic decision problems for *AVPA*, we introduce a new class of alternating automata called *alternating jump automata* (*AJA*). *AJA* operate on infinite words over a pushdown alphabet, are closed under boolean operations, and extend classical alternating *finite-state* automata by also allowing *non-local* moves. A non-local forward move leads a copy of the automaton from a call position to the matching-return position. We also allow local and non-local backward moves: by performing a non-local backward move, a copy of the automaton jumps from the current input position to the most recent unmatched call position. We show that one-way *AJA*

[1] The notion of alternating automaton over words or trees as defined in [12] applies to all known classes of nondeterministic automata such as pushdown automata or Turing machines.

and two-way *AJA* have the same expressiveness and capture exactly the class of *VPL*. Given a Büchi *NVPA* $\mathcal{P}$, one can construct an equivalent one-way Büchi *AJA* whose size is quadratic in the size of $\mathcal{P}$. Moreover, any parity two-way *AJA* $\mathcal{A}$ can be translated into an equivalent Büchi *NVPA* whose size is *singly* exponential in the size of $\mathcal{A}$. Some ideas in the proposed translation from two-way *AJA* to *NVPA* are taken from standard constructions for two-way finite-state automata [14,15]. However, due to the presence of both (local and non-local) forward and backward moves in two-way *AJA*, we have to face new non-trivial questions which require a more sophisticated approach.

Finally, we consider a fixpoint calculus, called *VP-μTL*, which subsumes CARET [3] and captures exactly the class of *VPL*. *VP-μTL* extends the full linear-time μ-calculus (with both forward and backward modalities) introduced in [14] by also allowing non-local forward and backward modalities corresponding to the abstract-next and caller modalities of CARET. We show that each *VP-μTL* sentence can be *linearly* translated into an equivalent parity two-way *AJA*, and vice versa. As a consequence satisfiability of *VP-μTL* is EXPTIME-complete and the pushdown model-checking problem against *VP-μTL* is EXPTIME-complete (and PTIME-complete in the size of the pushdown system), hence it is no more costly than that for weaker logics such as CARET. Note that the backward modalities in *VP-μTL* do not add any expressive power since future *VP-μTL* formulas correspond exactly to one-way *AJA*. However, many interesting properties which require for example inspection of the call-stack are much easier to express using past operators.

Due to the lack of space, for the omitted details we refer the interested reader to a forthcoming extended version of this paper.

Related work. As mentioned above, the class of *NVPA* over infinite words has been studied in [4]. In [4], it is also given a logical MSO-characterization of *VPL* and a characterization in terms of regular tree languages. Games on pushdown graphs against visibly pushdown winning conditions are decidable and have been studied in [11]. In [5], the results given in [4] are reformulated in terms of nondeterministic finite-state automata (*NFA*) over *nested words*. A nested word is an infinite word augmented with a binary relation over the set of positions which encodes the implicit nesting structure of calls and returns. An *NFA* over nested words behaves like an *NFA* over ordinary words with the difference that at a return, the next state depends on both the current state and the state at the matching call. In [2], the notion of nested word is extended to trees in order to allow the automata-theoretic specification of a class of branching-time context-free properties. In particular, the authors introduce one-way alternating automata over nested trees (*AP-NTA*): *AP-NTA* are strictly more expressive than their non-deterministic counterpart and while their emptiness is undecidable, the related pushdown model checking problem is instead EXPTIME-complete. Finally, an extension of the modal μ-calculus on nested trees as expressive as *AP-NTA* has been studied in [2,1]. When interpreted on infinite words over a pushdown alphabet, this logic corresponds exactly to future *VP-μTL*. As for the modal μ-calculus, the pushdown model checking problem for this new logic is EXPTIME-complete (even for a fixed formula). Satisfiability is instead undecidable.

2 Preliminaries

Labelled trees. Let $\mathbb{N}$ be the set of natural numbers. A tree T is a prefix closed subset of $\mathbb{N}^*$. Elements of T are called *nodes* and the empty word ε is the root of T. For $x \in T$, a child of x in T is a T-node of the form $x \cdot i$ with $i \in \mathbb{N}$. A *path* of T is a maximal sequence $x_0 x_1 \ldots$ of nodes s.t. $x_0 = \varepsilon$ and for each i, x_{i+1} is a child of x_i. For a set A, an A-labelled tree is a pair $r = \langle T, V \rangle$, where T is a tree and $V : T \rightarrow A$ maps each T-node to an element in A. For $x \in T$, *the subtree of r rooted at x* is the A-labelled tree $\langle T_x, V_x \rangle$, where $T_x = \{y \in \mathbb{N}^* \mid x \cdot y \in T\}$ and $V_x(y) = V(x \cdot y)$ for each $y \in T_x$.

Positive boolean formulas and regular acceptance conditions. Throughout this paper, we consider various classes of automata over infinite words equipped with parity or Büchi acceptance conditions over the finite set of (control) states. Formally, for a *finite* set Q, a parity condition over Q is a mapping $\Omega : Q \rightarrow \mathbb{N}$ assigning to each element in Q an integer (called *priority*). The *index* of Ω is the cardinality of the set $\{\Omega(q) \mid q \in Q\}$. A Büchi condition over Q is a subset F of Q. For an infinite sequence $\pi = q_0, q_1 \ldots$ over Q, we say that π satisfies the parity condition Ω if the smallest priority of the elements in Q that occur infinitely often along π is *even*. We say that π satisfies the Büchi condition F if there is some $q \in F$ that occurs infinitely often along π.

For a finite set X, $\mathcal{B}^+(X)$ denotes the set of positive boolean formulas over X built from elements in X using $\vee$ and $\wedge$ (we also allow the formulas `true` and `false`). A subset Y of X *satisfies* $\theta \in \mathcal{B}^+(X)$ iff the truth assignment that assigns `true` to the elements in Y and `false` to the elements of $X \setminus Y$ satisfies θ. The set Y *exactly satisfies* θ if Y satisfies θ and every proper subset of Y does not satisfy θ. The dual $\widetilde{\theta}$ of formula θ is obtained from θ by exchanging $\vee$ with $\wedge$ and `true` with `false`.

Visibly pushdown languages. A *pushdown alphabet* Σ is an alphabet which is partitioned in three disjoint finite alphabets Σ_c, Σ_r, and Σ_{int}, where Σ_c is a finite set of *calls*, Σ_r is a finite set of *returns*, and Σ_{int} is a finite set of *internal actions*.

A *Büchi nondeterministic visibly pushdown automaton* (Büchi *NVPA*) [4] on infinite words over a pushdown alphabet $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$ is a tuple $\mathcal{P} = \langle Q, q_0, \Gamma, \Delta, F \rangle$, where Q is a finite set of (control) states, $q_0 \in Q$ is the initial state, Γ is the finite stack alphabet, $\Delta \subseteq (Q \times \Sigma_c \times Q \times \Gamma) \cup (Q \times \Sigma_r \times (\Gamma \cup \{\perp\}) \times Q) \cup (Q \times \Sigma_{int} \times Q)$ is the transition relation (where $\perp \notin \Gamma$ is the special *stack bottom symbol*), and $F \subseteq Q$ is a Büchi condition over Q. A transition of the form $(q, a, q', B) \in Q \times \Sigma_c \times Q \times \Gamma$ is a push transition, where on reading the call a the symbol $B \neq \perp$ is pushed onto the stack and the control changes from q to q'. A transition of the form $(q, a, B, q') \in Q \times \Sigma_r \times (\Gamma \cup \{\perp\}) \times Q$ is a pop transition, where on reading the return a, B is popped from the stack and the control goes from q to q'. Finally, on reading an internal action a, $\mathcal{P}$ can choose only transitions of the form (q, a, q') which do not use the stack. Thus, $\mathcal{P}$ pushes onto the stack only on reading a call, pops the stack only at returns, and does not use the stack on internal actions. Hence, the input controls the kind of operations permissible on the stack, and thus the stack depth at every position [4].

A configuration of $\mathcal{P}$ is a pair (q, β), where $q \in Q$ and $\beta \in \Gamma^* \cdot \{\perp\}$ is a stack content. For $w \in \Sigma^\omega$, $w(i)$ denotes the i-th symbol of w. A run of $\mathcal{P}$ over w is an infinite sequence of configurations $r = (q'_0, \beta_0)(q'_1, \beta_1) \ldots$ such that $\beta_0 = \perp$, q'_0 is the initial state, and for each $i \geq 0$: [**push**] if $w(i)$ is a call, then $\exists B \in \Gamma$ such that $\beta_{i+1} = B \cdot \beta_i$ and $(q'_i, w(i), q'_{i+1}, B) \in \Delta$; [**pop**] if $w(i)$ is a return, then $\exists B \in \Gamma$ s.t. $(q'_i, w(i), B, q'_{i+1}) \in \Delta$ and *either* $\beta_i = \beta_{i+1} = B = \bot$, or $B \neq \bot$ and $\beta_i = B \cdot \beta_{i+1}$; [**internal**] if $w(i)$ is an internal action, $(q'_i, w(i), q'_{i+1}) \in \Delta$ and $\beta_i = \beta_{i+1}$. The run is accepting iff its projection over Q satisfies the Büchi condition F. The language of $\mathcal{P}$, $\mathcal{L}(\mathcal{P})$, is the set of $w \in \Sigma^\omega$ s.t. there is an accepting run of $\mathcal{P}$ over w. A language $\mathcal{L}$ over Σ is a *visibly pushdown language* (*VPL*) if there is a Büchi *NVPA* $\mathcal{P}$ s.t. $\mathcal{L}(\mathcal{P}) = \mathcal{L}$.

In order to model formal verification problems of pushdown systems M using finite specifications (such as *NVPA*) denoting *VPL* languages, we choose a suitable pushdown alphabet $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$, and associate a symbol in Σ with each transition of M with the restriction that push transitions are mapped to Σ_c, pop transitions are mapped to Σ_r, and transitions that do not use the stack are mapped to Σ_{int}. Note that M equipped with such a labelling is a Büchi *NVPA* where all the states are accepting. The specification S describes another *VPL* $\mathcal{L}(S)$ over the same alphabet, and M is correct iff $\mathcal{L}(M) \subseteq \mathcal{L}(S)$. Given a class $\mathcal{C}$ of specifications describing *VPL* over Σ, the *pushdown model checking problem against* $\mathcal{C}$*-specifications* is to decide, given a pushdown system M over Σ and a specification S in the class $\mathcal{C}$, whether $\mathcal{L}(M) \subseteq \mathcal{L}(S)$.

3 Alternating Visibly Pushdown Automata

In this section we study the class of *alternating visibly pushdown automata* (*AVPA*). We show that any parity *AVPA* $\mathcal{P}$ can be translated into an equivalent Büchi *NVPA* whose size is doubly exponential in the size of $\mathcal{P}$. This double-exponential blowup cannot be avoided. In fact we show that emptiness for this class of automata is 2EXPTIME-complete (recall that emptiness for parity or Büchi *NVPA* is in PTIME [4]).

As *NVPA*, an *AVPA* $\mathcal{P}$ pushes onto (resp., pops) the stack only when it reads a call (resp., a return), and does not use the stack on internal actions. However, at any instant $\mathcal{P}$ can choose nondeterministically to split in *many* copies, each of them moving to the next input symbol. Formally, a parity *AVPA* over a pushdown alphabet Σ is a tuple $\mathcal{P} = \langle Q, q_0, \Gamma, \delta, \Omega \rangle$, where Q, q_0, and Γ are defined as for *NVPA*, Ω is a parity condition over Q, and $\delta : Q \times \Sigma \times (\Gamma \cup \{\bot\}) \to \mathcal{B}^+(Q) \cup \mathcal{B}^+(Q \times \Gamma)$ is the transition function satisfying: (i) $\delta(q, a, B) \in \mathcal{B}^+(Q)$ if a is *not* a call, (ii) $\delta(q, a, B) \in \mathcal{B}^+(Q \times \Gamma)$ if a is a call, and (3) $\delta(q, a, B) = \delta(q, a, B')$ if a is *not* a return.

Given a word $w \in \Sigma^* \cup \Sigma^\omega$, a state q, and stack content $\beta \in \Gamma^* \cdot \{\bot\}$, a (q, β)-*run of* $\mathcal{P}$ *over* w is a $Q \times \Gamma^* \cdot \{\bot\}$-labelled tree $r = \langle T, V \rangle$, where each node x of T labelled by (q', β') describes a copy of $\mathcal{P}$ in state q' and stack content β' reading the symbol $w(|x|)$. Moreover, we require that $V(\varepsilon) = (q, \beta)$ and for each $x \in T$ with $V(x) = (q', B \cdot \beta')$, there is a set $H = \{p_0, \ldots, p_m\}$ exactly satisfying $\delta(q', w(|x|), B)$ (note that $H \subseteq Q \times \Gamma$ if $w(|x|)$ is a call, and $H \subseteq Q$ otherwise) such that x has children $x \cdot 0, \ldots, x \cdot m$ and for all $0 \leq i \leq m$, the following holds: [**Push**] If $w(|x|)$ is a call, $p_i = (q_i, B_i)$ and $V(x \cdot i) = (q_i, B_i \cdot B \cdot \beta')$; [**Pop**] If $w(|x|)$ is a return, $V(x \cdot i) = (p_i, \beta')$ if $B \neq \top$, and $V(x \cdot i) = (p_i, \bot)$ otherwise; [**Internal**] If $w(|x|)$ is an internal action, $V(x \cdot i) = (p_i, B \cdot \beta')$. The run $r = \langle T, V \rangle$ is *memoryless* if for all $x_1, x_2 \in T$ such that $|x_1| = |x_2|$ and $V(x_1) = V(x_2)$, the subtrees of r rooted at x_1 and x_2 coincide (i.e., fixed a position along w, the behaviour of $\mathcal{P}$ depends only on the current state and stack content, and is independent on the past choices). If w is infinite, the run r is *accepting* if for each infinite path $\pi = x_0 x_1 \ldots$ of r, the projection of $V(x_0)V(x_1)\ldots$ over Q satisfies the parity condition Ω. The ω-language $\mathcal{L}(\mathcal{P})$ of $\mathcal{P}$ is the set of $w \in \Sigma^\omega$ such that there is an accepting $(q_0, \bot)$-run of $\mathcal{P}$ over w.

Remark 1. For $w \in \Sigma^\omega$, we can associate in a standard way [12] to $\mathcal{P}$ and w an infinite-state parity game, where player 0 plays for acceptance, while player 1 plays for rejection. Winning strategies of player 0 correspond to accepting runs of $\mathcal{P}$ over w. Since the existence of a winning strategy in parity games implies the existence of a memoryless one, we can restrict ourselves to consider only memoryless runs of $\mathcal{P}$. Moreover, by [12] the *dual automaton* (*AVPA*) $\widetilde{\mathcal{P}} = \langle Q, q_0, \Gamma, \widetilde{\delta}, \widetilde{\Omega}\rangle$ of $\mathcal{P}$, where $\widetilde{\delta}(q, a, B)$ is the dual of $\delta(q, a, B)$ and $\widetilde{\Omega}(q) = \Omega(q) + 1$ for all $q \in Q$, accepts the complement of $\mathcal{L}(\mathcal{P})$.

From parity *AVPA* to Büchi *NVPA*. Fix a parity *AVPA* $\mathcal{P} = \langle Q, q_0, \Gamma, \delta, \Omega\rangle$ over Σ. Let n be the index of Ω and $[n] = \{0, \ldots, n\}$. W.l.o.g. assume that for each $q \in Q$, $0 \leq \Omega(q) \leq n$ and $\delta(q, a, B) \notin \{\mathtt{true}, \mathtt{false}\}$. We will construct a Büchi *NVPA* accepting $\mathcal{L}(\mathcal{P})$. Our approach is a generalization of the technique of "summaries" used in [4] to show that *NVPA* are closed under complementation.

A finite word $w \in \Sigma^*$ is *well-matched* if inductively or (1) $w = \epsilon$, or (2) $w = aw'$, $a \in \Sigma_{int}$ and w' is well-matched, or (3) $w = a_c w' a_r w''$, $a_c \in \Sigma_c$, $a_r \in \Sigma_r$, and w' and w'' are well-matched. The set L_{mwm} of *minimally well-matched words* is the set of words of the form $a_c w a_r$ where $a_c \in \Sigma_c$, $a_r \in \Sigma_r$, and w is well-matched.

Let $P \subseteq Q \times [n]$. For a *finite* run $r = \langle T, V\rangle$ of $\mathcal{P}$ over $w \in \Sigma^*$, r is *consistent with* P if for each $(q, i) \in Q \times [n]$: $(q, i) \in P$ iff there is a path $\pi = x_0 \ldots x_k$ (note that $k = |w|$) of r with $V(x_j) = (q_j, \beta_j)$ for all $0 \leq j \leq k$ such that $q_k = q$ and $i = min\{\Omega(q_j)\}_{0 \leq j \leq k}$.

Let $\mathcal{H} = Q \times 2^{Q \times [n]}$. A *summary* for a well-matched word $w \in \Sigma^*$ is a nonempty set $\mathcal{S} \subseteq \mathcal{H}$ such that for all $(q, P) \in \mathcal{S}$, there is a memoryless (q, β)-run of $\mathcal{P}$ over w for some stack content β which is consistent with P. Intuitively, each $(q, P) \in \mathcal{S}$ keeps track of the meaningful information associated with some (q, β)-run r of $\mathcal{P}$ over w; in particular, for each path π of r, P keeps track of the smallest priority of the states visited by π and of the last state along π (since w is well-matched, the stack content associated with such a state is still β and the initial stack content β is not modified along the run r). Given a memoryless run r of $\mathcal{A}$ over an *infinite* word w', the notion of summary is used to capture the meaningful information of the portions of r associated with well-matched subwords w of w'. Note that for such a subword w, fixed a state q, there can be different portions of r associated with w corresponding to finite (q, β)-runs of $\mathcal{P}$ over w whose initial stack contents are distinct. Since the local choices of $\mathcal{P}$ depend also on the stack content, a summary must keep track for each state q of distinct sets $P_1, \ldots, P_n \subseteq Q \times [n]$, which are consistent with different finite (q, β)-runs over w.

For $w \in \Sigma^\omega$, there is a unique factorization $w_1 w_2 \ldots$ of w such that for all $i \geq 1$, $w_i \in \Sigma \cup L_{mwm}$ and if w_i is a call, then there is no $j > i$ such that w_j is a return. Let $\widehat{\Sigma}$ be the pushdown alphabet given by $\Sigma \cup (2^{\mathcal{H}} \setminus \{\emptyset\})$, where symbols in $2^{\mathcal{H}} \setminus \{\emptyset\}$ are internal actions. A *pseudo-word* $\widehat{w}$ is an infinite word over $\widehat{\Sigma}$ s.t. if $\widehat{w}(i)$ is a call, then there is no $j > i$ such that $\widehat{w}(j)$ is a return. Given $w \in \Sigma^\omega$, a *pseudo-word of*

w is obtained by replacing each factor $w_i \in L_{mwm}$ in the factorization of w with a summary of w_i.

Let us consider the *AVPA* $\widehat{\mathcal{P}} = \langle Q \times \{1, \ldots, n\}, (q_0, \Omega(q_0)), \Gamma, \widehat{\delta}, \widehat{\Omega} \rangle$ over $\widehat{\Sigma}$ where (1) for all $a \in \Sigma$, $\widehat{\delta}((q,i), a, B)$ is obtained from $\delta(q, a, B)$ by replacing each $q' \in Q$ with $(q', \Omega(q'))$, (2) for all $\mathcal{S} \in 2^{\mathcal{H}} \setminus \{\emptyset\}$, $\widehat{\delta}((q,i), \mathcal{S}, B) = \bigvee_{(q,P) \in \mathcal{S}} \bigwedge_{(p,h) \in P} (p, h)$, and (3) $\widehat{\Omega}((q,i)) = i$. For a word $\widehat{w} \in \widehat{\Sigma}^\omega$, $\widehat{\mathcal{P}}$ simulates $\mathcal{P}$ over the Σ-letters of $\widehat{w}$, and on letters $\mathcal{S} \in 2^{\mathcal{H}} \setminus \{\emptyset\}$ it updates the current state q by splitting in n copies in states $(q_1, i_1), \ldots, (q_n, i_n)$, respectively, such that $(q, \{(q_1, i_1), \ldots, (q_n, i_n)\}) \in \mathcal{S}$. By construction for every $w \in \Sigma^\omega$, $\mathcal{P}$ accepts w iff there is a pseudo-word of w that is accepted by $\widehat{\mathcal{P}}$. Note that for a $((q_0, \Omega(q_0)), \bot)$-run of $\widehat{\mathcal{P}}$ over a pseudo-word, whenever a return occurs, the current stack content is empty. Hence, we can construct a parity alternating *finite-state* automaton $\mathcal{A}$ that simulates $\widehat{\mathcal{P}}$ over pseudo-words and accepts only pseudo-words (on reading a return, $\mathcal{A}$ simulates $\widehat{\mathcal{P}}$ when the stack is empty). By [13,15] one can construct a nondeterministic Büchi finite-state automaton $\mathcal{A}_{PW}$ with a number of states exponential in $|Q|$ that accepts $\mathcal{L}(\mathcal{A})$. Thus, we obtain the following:

Proposition 1. *One can construct a nondeterministic Büchi finite-state automaton $\mathcal{A}_{PW}$ over $\widehat{\Sigma}$ whose number of states is exponential in the number of states of $\mathcal{P}$ such that for each $w \in \Sigma^\omega$: $w \in \mathcal{L}(\mathcal{P})$ iff there is a pseudo-word of w that is accepted by $\mathcal{A}_{PW}$.*

The next step consists of computing the summary information associated with minimally well-matched words.

Proposition 2. *We can build in* doubly exponential *time a* NVPA $\mathcal{P}_S$ *on* finite *words over Σ with a special stack symbol '$-$' and set of states containing $2^{\mathcal{H}}$ such that for each $w \in \Sigma^*$, $\mathcal{S} \in 2^{\mathcal{H}} \setminus \{\emptyset\}$, and stack content $\beta \in \{-\}^*.\bot$: there is an accepting $(\mathcal{S}, \beta)$-run of $\mathcal{P}_S$ over w* iff *$w \in L_{mwm}$ and $\mathcal{S}$ is a summary of w. $\mathcal{P}_S$ has a unique accepting state $q_{acc} \notin 2^{\mathcal{H}}$ (with no outgoing transitions).*

Now, we are ready to construct a Büchi *NVPA* $\mathcal{P}_N$ accepting exactly $\mathcal{L}(\mathcal{P})$. Let $\mathcal{A}_{PW}$ be the Büchi nondeterministic finite-state automaton of Proposition 1 with states Q_{PW}, and let $\mathcal{P}_S$ be the *NVPA* of Proposition 2 with states $Q_S \supseteq 2^{\mathcal{H}}$, accepting state $q_{acc} \notin 2^{\mathcal{H}}$, and special stack symbol '$-$'. Given a word $w \in \Sigma^\omega$, $\mathcal{P}_N$ guesses a pseudo word $\widehat{w}$ of w and checks that it is in $\mathcal{L}(\mathcal{A}_{PW})$. The set of states of $\mathcal{P}_N$ is $Q_{PW} \times Q_S$. At any step, $\mathcal{P}_N$ either simulates $\mathcal{A}_{PW}$ on the first component of the state (and the other one remains constant with value q_{acc}) pushing onto the stack only the symbol '$-$', or simulates $\mathcal{P}_S$ on the second component (and the other one remains constant). Whenever a call a_c occurs and $\mathcal{P}_N$ is in state (q^1_{PW}, q_{acc}), $\mathcal{P}_N$ can guess that a_c is the first letter of a factor $w_m \in L_{mwm}$ of w and there is a summary $\mathcal{S} \in 2^{\mathcal{H}}$ of w_m by simulating a push move of $\mathcal{P}_S$ with source state $\mathcal{S}$, input symbol a_c, and target q_S, and moving to state (q^2_{PW}, q_S) s.t. $q^1_{PW} \xrightarrow{\mathcal{S}} q^2_{PW}$ is a transition of $\mathcal{A}_{PW}$ (by Proposition 2, $q_S \neq q_{acc}$). From (q^2_{PW}, q_S), $\mathcal{P}_N$ simulates $\mathcal{P}_S$. If the guess was correct, then $\mathcal{P}_N$ can reach a state of the form (q^2_{PW}, q_{acc}), and the whole procedure is repeated. Otherwise, $\mathcal{P}_N$ will continue to simulate $\mathcal{P}_S$ visiting only states of the form (q, q') with $q' \neq q_{acc}$. Therefore, the accepting states of $\mathcal{P}_N$ are of the form (q, q_{acc}) where q is an accepting state of $\mathcal{A}_{PW}$. Thus, we obtain the following:

Theorem 1. *Given a parity* AVPA $\mathcal{P}$*, one can construct in* doubly exponential *time a Büchi* NVPA $\mathcal{P}_N$ *with size doubly exponential in the size of* $\mathcal{P}$ *s.t.* $\mathcal{L}(\mathcal{P}_N) = \mathcal{L}(\mathcal{P})$.

***Decision problems for (parity)* AVPA.** First, we consider emptiness and universality. Note that these problems are equivalent from the complexity point of view since the dual automaton of a *AVPA* $\mathcal{P}$ has size linear in the size of $\mathcal{P}$. Since emptiness of Büchi *NVPA* is in PTIME, by Theorem 1, emptiness of *AVPA* is in 2EXPTIME. We can show that the problem is also 2EXPTIME-hard (also for Büchi *AVPA*) by a reduction from the word problem for EXPSPACE-bounded Turing Machines. For the pushdown model checking problem, given a pushdown system M and an *AVPA* $\mathcal{P}$, checking whether $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{P})$ reduces to checking emptiness of $\mathcal{L}(M) \cap \mathcal{L}(\widetilde{\mathcal{P}}_N)$, where $\mathcal{P}_N$ is the *NVPA* equivalent to the dual of $\mathcal{P}$. By [4] and Theorem 1 this check can be done in time polynomial in the size of M and doubly exponential in the size of $\mathcal{P}$. The problem is at least as hard as universality of *AVPA*. Thus, we obtain the following:

Theorem 2. *Emptiness and universality of* (*parity or Büchi*) AVPA *are* 2EXPTIME-*complete. Moreover, the pushdown model checking problem against* AVPA *specifications is* 2EXPTIME-*complete (and polynomial in the size of the pushdown system).*

4 Alternating Jump Finite-State Automata

In this section we introduce the class of *alternating jump* (*finite-state*) *automata* (*AJA*) operating on infinite words over a pushdown alphabet. *AJA* extend standard alternating finite-state automata by also allowing non-local moves: when the current input symbol is a call a_c and the *matching return* a_r of a_c along the input word exists, a copy of the automaton can move (jump) to the return a_r. We also allow local and non-local backward moves (details are given below). We show that one-way and two-way *AJA* have the same expressiveness and capture exactly the class of visibly pushdown languages. The main result is an algorithm for translating a given parity two-way *AJA* (*2-AJA*) $\mathcal{A}$ into an equivalent Büchi *NVPA* whose size is *singly* exponential in the size of $\mathcal{A}$. We also study some decision problems for the considered class of automata.

Fix a pushdown alphabet Σ. Given an infinite word $w \in \Sigma^\omega$, we consider four different notions of successor for a position $i \in \mathbb{N}$ along w:

- The *forward local successor of* i *along* w, written $succ(\downarrow, w, i)$, is $i + 1$.
- The *backward local successor of* i *along* w, written $succ(\uparrow, w, i)$, is $i - 1$ if $i > 0$, and it is undefined otherwise (in this case we set $succ(\uparrow, w, i) = \top$).
- The *abstract successor of* i *along* w [3], written $succ(\downarrow_{\mathsf{a}}, w, i)$, is the forward local successor if i is not a call position. If instead $w(i)$ is a call, $succ(\downarrow_{\mathsf{a}}, w, i)$ points to the *matching return position* of i (if any), i.e.: if there is $j > i$ such that $w(j)$ is a return and $w(i + 1) \dots w(j - 1)$ is well-matched, then $succ(\downarrow_{\mathsf{a}}, w, i) = j$ (note that j is uniquely determined), otherwise $succ(\downarrow_{\mathsf{a}}, w, i) = \top$.
- The *caller of* i *along* w [3], written $succ(\uparrow^{\mathsf{c}}, w, i)$, points to the last unmatched call of the prefix of w until position i. Formally, if there is $j < i$ such that $w(j)$ is a call and $w(j + 1) \dots w(h)$ is well-matched (where $h = i - 1$ if i is a call position, and $h = i$ otherwise), then $succ(\uparrow^{\mathsf{c}}, w, i) = j$ (note that j is uniquely determined), otherwise the caller is undefined and we set $succ(\uparrow^{\mathsf{c}}, w, i) = \top$.

Let $DIR = \{\downarrow, \downarrow_{\mathsf{a}}, \uparrow, \uparrow^{\mathsf{c}}\}$. A *parity 2-AJA* over Σ is a tuple $\mathcal{A} = \langle Q, q_0, \delta, \Omega\rangle$, where Q, q_0, and Ω are defined as for parity *AVPA* and $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(DIR \times Q \times Q)$ is the transition function. Intuitively, a target of a move of $\mathcal{A}$ is encoded by a triple $(dir, q, q') \in DIR \times Q \times Q$, meaning that a copy of $\mathcal{A}$ moves to the dir-successor of the current input position i in state q if such a successor is defined, and to position $i + 1$ in state q' otherwise. Note that the q'-component of the triple above is irrelevant if $dir = \downarrow$ (we give it only to have a uniform notation). A *1-AJA* is a *2-AJA* whose transition function δ satisfies $\delta(q, a) \in \mathcal{B}^+(\{\downarrow, \downarrow_{\mathsf{a}}\} \times Q \times Q)$ for each $(q, a) \in Q \times \Sigma$.

A q-run of $\mathcal{A}$ over an infinite word $w \in \Sigma^\omega$ is a $\mathbb{N} \times Q$-labelled tree $r = \langle T, V\rangle$, where a node $x \in T$ labelled by (i, q') describes a copy of $\mathcal{A}$ that is in q' and reads the i-th input symbol. Moreover, we require that $r(\varepsilon) = (0, q)$ and for all $x \in T$ with $r(x) = (i, q')$, there is a set $H = \{(dir_0, q'_0, q''_0), \ldots, (dir_m, q'_m, q''_m)\} \subseteq DIR \times Q \times Q$ exactly satisfying $\delta(q', w(i))$ such that the children of x are $x \cdot 0, \ldots, x \cdot m$, and for each $0 \leq h \leq m$: $V(x \cdot h) = (i + 1, q''_h)$ if $succ(dir_h, w, i) = \top$, and $V(x \cdot h) = (succ(dir_h, w, i), q'_h)$ otherwise. The run r is *memoryless* if for all nodes $x, y \in T$ such that $V(x) = V(y)$, the (labelled) subtrees rooted at x and y coincide. The q-run r is *accepting* if for each infinite path $x_0 x_1 \ldots$, the projection over Q of $V(x_0)V(x_1)\ldots$ satisfies the parity condition Ω. The ω-language of $\mathcal{A}$, $\mathcal{L}(\mathcal{A})$, is the set of words $w \in \Sigma^\omega$ such that there is an accepting q_0-run r of $\mathcal{A}$ over w. A q_0-run is called simply run.

As for *AVPA*, we can give a standard game-theoretic interpretation of acceptance in *AJA*. In particular, by [12] the *dual automaton* $\widetilde{\mathcal{A}} = \langle Q, q_0, \widetilde{\delta}, \widetilde{\Omega}\rangle$ of an *AJA* $\mathcal{A}$, where for all $(q, a) \in Q \times \Sigma$, $\widetilde{\Omega}(q) = \Omega(q) + 1$ and $\widetilde{\delta}(q, a)$ is the dual of $\delta(q, a)$, accepts the complement of $\mathcal{L}(\mathcal{A})$. Moreover, the existence of an accepting run of $\mathcal{A}$ over w implies the existence of a memoryless one. Since *AJA* are clearly closed under intersection and union, we obtain the following result.

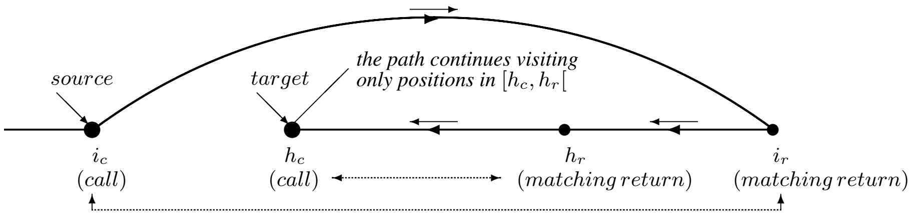
Proposition 3. 2-AJA *and* 1-AJA *are closed under boolean operations. Moreover, given a* 2-AJA $\mathcal{A}$, $w \in \mathcal{L}(\mathcal{A})$ *iff there is an accepting memoryless run of* $\mathcal{A}$ *over* w.

4.1 Relation Between Nondeterministic Visibly Pushdown Automata and *AJA*

In this subsection we present translations forth and back between *NVPA* and *2-AJA*.

***From parity* 2-AJA *to Büchi* NVPA.** Fix a parity *2-AJA* $\mathcal{A} = \langle Q, q_0, \delta, \Omega\rangle$ over Σ. Let n be the index of Ω and $[n] = \{0, \ldots, n\}$. Without loss of generality we can assume that for each $q \in Q$, $0 \leq \Omega(q) \leq n$ and $\delta(q, a) \notin \{\texttt{true}, \texttt{false}\}$.

By Proposition 3, we can restrict ourselves to consider *memoryless* runs of $\mathcal{A}$. For a word $w \in \Sigma^\omega$, we represent memoryless runs of $\mathcal{A}$ over w as follows. For a set $H \subseteq Q \times DIR \times Q \times Q$, let $Dom(H) = \{q \in Q \mid (q, dir, q', q'') \in H\}$. A *strategy of $\mathcal{A}$ over w* is a mapping $\mathsf{St} : \mathbb{N} \rightarrow 2^{Q \times DIR \times Q \times Q}$ satisfying: (i) $q_0 \in Dom(\mathsf{St}(0))$, (ii) for each $i \in \mathbb{N}$ and $q \in Dom(\mathsf{St}(i))$, the set $\{(dir, q', q'') \mid (q, dir, q', q'') \in \mathsf{St}(i)\}$ exactly satisfies $\delta(q, w(i))$, and (iii) for each $(q, dir, q', q'') \in \mathsf{St}(i)$, $q' \in Dom(\mathsf{St}(h))$ if $succ(dir, w, i) = h \neq \top$, and $q'' \in Dom(\mathsf{St}(i + 1))$ otherwise. Intuitively, St is an infinite word over $2^{Q \times DIR \times Q \times Q}$ encoding a memoryless run r of $\mathcal{A}$ over w. In particular, $Dom(\mathsf{St}(i))$ is the set of states in which the automaton is (along r) when $w(i)$ is read, and for $q \in Dom(\mathsf{St}(i))$, the set $\{(dir, q', q'') \mid (q, dir, q', q'') \in \mathsf{St}(i)\}$ is the set of choices made by $\mathcal{A}$ on reading $w(i)$ in state q.



source
target
the path continues visiting only positions in $[h_c, h_r[$
i_c $(call)$
h_c $(call)$
h_r $(matching\,return)$
i_r $(matching\,return)$

Fig. 1. Structure of a zig-zag prefix

A j-path γ of St is a sequence of pairs in $\mathbb{N} \times Q$ of the form $\gamma = (i_1, q_1)(i_2, q_2)\ldots$ where $i_1 = j$ and for each $1 \leq h < |\gamma|$: $\exists(q_h, dir, q'_h, q''_h) \in \mathsf{St}(i_h)$ such that *either* $i_{h+1} = succ(dir, w, i_h)$ and $q_{h+1} = q'_h$, *or* $succ(dir, w, i_h) = \top$, $i_{h+1} = i_h + 1$ and $q_{h+1} = q''_h$. The path γ is *forward* (resp., *backward*) if $i_{h+1} > i_h$ (resp., $i_{h+1} < i_h$) for any h. For a finite path $\gamma = (i_1, q_1)\ldots(i_k, q_k)$, the *index of* γ is $min\{\Omega(q_l) \mid 1 \leq l \leq k\}$ if $k > 1$, and $n + 1$ otherwise. In case $i_k = i_1$, we say that γ is *closed*. For a finite path $\gamma_1 = (i_1, q_1), \ldots, (i_k, q_k)$ and a path $\gamma_2 = (i_k, q_k)(i_{k+1}, q_{k+1})\ldots$, let $\gamma_1 \circ \gamma_2$ be the path $\gamma_1(i_{k+1}, q_{k+1})\ldots$. A path γ is *positive* (resp., *negative*) if γ is of the form $\gamma = \gamma_1 \circ \gamma_2 \circ \ldots$, where each γ_i is either a closed path, or a forward (resp., backward) path. A i-*cycle* of St is an infinite path γ that can be decomposed in the form $\gamma = \gamma_1 \circ \gamma_2 \circ \ldots$, where each γ_h is a closed i-path.

The strategy St is *accepting* if for each infinite path γ starting from $(0, q_0)$, the projection of γ over Q satisfies the parity condition Ω of $\mathcal{A}$. Clearly, there is an accepting memoryless run of $\mathcal{A}$ over w iff there is an accepting strategy of $\mathcal{A}$ over w.

Now, we have to face the problem that the infinite paths γ of St can use backward moves. If γ is *positive*, then the idea is to collapse the closed subpaths (in the decomposition of γ) in a unique move and to keep track of the associated meaningful information by finite local auxiliary structures. However, there can be infinite paths that are not positive. Fortunately, also for this class of paths, it is possible to individuate a decomposition that is convenient for our purposes. This decomposition is formalized as follows. A *zig-zag i-path* is an infinite i-path γ inductively defined as follows:

– (**Positive prefix**) $\gamma = \gamma_p \circ \gamma_s$, where γ_p is a finite *non-empty* positive i-path leading to position $h \geq i$, and γ_s is a zig-zag h-path visiting only positions in $[h, \infty[$.
– (**Zig-zag prefix**) $w(i)$ is a call and $\gamma = \gamma_{i,i_r} \circ \gamma_{i_r,h_r} \circ \gamma_{h_r,h_c} \circ \gamma_s$, where γ_{i,i_r} is a path from i to the matching return position i_r, γ_{i_r,h_r} is a *negative* path from i_r to the return position $h_r \in]i, i_r[$, γ_{h_r,h_c} is a path from h_r to the matching-call position $h_c \in]i, i_r[$, and γ_s is a zig-zag h_c-path visiting only positions in $[h_c, h_r[$.
– (**terminal Zig-zag**): $\gamma = \gamma_p \circ \gamma_s$, where γ_s is a h-cycle, and *either* γ_p is empty and $h = i$, *or* $\gamma_p = \gamma_{i,i_r} \circ \gamma_{i_r,h}$, where: γ_{i,i_r} is a path from the call position i to the matching return position i_r, and $\gamma_{i_r,h}$ is a negative path from i_r to $h \in]i, i_r[$.

The structure of a zig-zag prefix is illustrated in Figure 1. Note that an infinite positive path is a special case of zig-zag path. The following holds.

Proposition 4. *An infinite i-path visiting only positions in $[i, \infty[$ is a* zig-zag i-path.

Our next goal is to keep track locally for each position i of the meaningful information associated with closed i-paths. In case $w(i)$ is a return with matching-call $w(i_c)$, we also need to keep track in a finite way of the i_c-paths (resp., i-paths) leading to i (resp., i_c). Thus, we give the following definition. A *path summary of* $\mathcal{A}$ is a triple of mappings $(\Delta, \Delta_{\mathsf{a}}, \Delta_{\mathsf{c}})$ where $\Delta : \mathbb{N} \rightarrow 2^{Q\times[n+1]\times Q}$ and $\Delta_{\mathsf{a}}, \Delta_{\mathsf{c}} : \mathbb{N} \times \{\mathsf{down}, \mathsf{up}\} \rightarrow 2^{Q\times[n+1]\times Q}$. Such a triple is a *path summary of the strategy* St if it satisfies some *closure conditions*. Since there are many conditions, we do not formalize them here. We only give their general form, to show (as we will see) that they can be checked by an *NVPA* (using the stack): each condition has the form $check(i, h) \implies \Phi(i, h)$, where (1) $i, h \in \mathbb{N}$ are implicitly universally quantified, (3) $check(i, h) := succ(dir, w, i) = h \mid succ(dir, w, i) = \top \wedge h = i + 1$ (where $dir \in DIR$). (2) $\Phi(i, h)$ is a boolean formula over propositions of the form $(q, dir, q', q'') \in \mathsf{St}(j)$ or $(q, in, q') \in \mathcal{F}(j)$, where $j \in \{i, h\}$, and $\mathcal{F}(j) \in \{\Delta(j), \Delta_{\mathsf{a}}(j, \mathsf{down}), \Delta_{\mathsf{a}}(j, \mathsf{up}), \Delta_{\mathsf{c}}(j, \mathsf{down}), \Delta_{\mathsf{c}}(j, \mathsf{up})\}$.

The *intended meaning* for a path summary $(\Delta, \Delta_{\mathsf{a}}, \Delta_{\mathsf{c}})$ of strategy St is as follows:

- $(q, in, q') \in \Delta(i)$ **if** there is a closed i-path of St of index in from (i, q) to (i, q').
- $(q, in, q') \in \Delta_{\mathsf{a}}(i, \mathsf{up})$ (resp., $(q, in, q') \in \Delta_{\mathsf{a}}(i, \mathsf{down})$) **if** $i = succ(\downarrow_{\mathsf{a}}, w, h)$, $w(h)$ is a call, and there is a path of index in from (i, q) to (h, q') (resp., from (h, q) to (i, q')).
- $(q, in, q') \in \Delta_{\mathsf{c}}(i, \mathsf{up})$ (resp., $(q, in, q') \in \Delta_{\mathsf{c}}(i, \mathsf{down})$) **if** $succ(\uparrow^{\mathsf{c}}, w, i) = h$ and there is a path of of index in from (i, q) to (h, q') (resp., from (h, q) to (i, q')).

By using the path summary $(\Delta, \Delta_{\mathsf{a}}, \Delta_{\mathsf{c}})$ of St, we define a convenient representation for zig-zag paths which can be simulated by *NVPA*. A forward (resp., backward) move of St and $(\Delta, \Delta_{\mathsf{a}}, \Delta_{\mathsf{c}})$ is a pair $(i_1, q_1, in_1) \rightarrow (i_2, q_2, in_2)$ of triples in $\mathbb{N} \times Q \times [n+1]$ such that $i_2 > i_1$ (resp., $i_1 > i_2$) and: $\exists (q_1, in, q) \in \Delta(i_1)$ such that $(i_1, q)(i_2, q_2)$ is a path of St and $in_1 = min\{in, \Omega(q)\}$. A *downward path* ρ *of* St *and* $(\Delta, \Delta_{\mathsf{a}}, \Delta_{\mathsf{c}})$ is a sequence $\rho = (i_1, q_1, in_1)(i_2, q_2, in_2)\ (i_3, q_3, in_3) \ldots$ inductively defined as follows:

- (**Forward move**) $(i_1, q_1, in_1) \rightarrow (i_2, q_2, in_2)$ is a forward move and (i_2, q_2, in_2) $(i_3, q_3, in_3) \ldots$ is a downward path visiting only positions in $[i_2, \infty[$.
- (**Zig-zag move**) $i_1 < i_2$, $w(i_1)$ and $w(i_2)$ are calls with matching-return positions h_1^r and h_2^r, there is a path $(h_1^r, q_1^r, in_1^r) \ldots (h_2^r, q_2^r, in_2^r)$ using only backward moves s.t. $(q_1, in, q_1^r) \in \Delta_{\mathsf{a}}(h_1^r, \mathsf{down})$ and $(q_2^r, in', q_2) \in \Delta_{\mathsf{a}}(h_2^r, \mathsf{up})$ for some $in, in' \in [n]$, and $\rho' = (i_2, q_2, in_2)\ (i_3, q_3, in_3) \ldots$ is a downward path visiting only positions in $[i_2, h_r^2[$.
- (**Terminal move**) $\rho = \rho'(i, q, in)(i, q', in')$, where $(q, in, q') \in \Delta(i)$, $(q', in', q') \in \Delta(i)$, and $in' \in [n]$. Moreover, either ρ' is empty or $\rho' = (i_c, q_c, in_c)$, i_c is a call position with matching-return i_r, $i \in]i_c, i_r[$, and there are $(q_c, in_c, q_r) \in \Delta_{\mathsf{a}}(i_r, \mathsf{down})$ and a path $(i_r, q_r, in_r) \ldots (i, q, in)$ using only backward moves.

Note that a downward path is either infinite (and uses only forward moves) or is finite and ends with a terminal move (corresponding to a cycle of St). Let Ω' be the parity condition over $Q \times [n + 1]$ defined as $\Omega'((q, in)) = in$. A downward path ρ is *accepting* if either (i) ρ is infinite and its projection over $Q \times [n + 1]$ satisfies Ω' or (ii) ρ is finite and leads to a triple (i, q, in) such that in is *even*. The path summary $(\Delta, \Delta_{\mathsf{a}}, \Delta_{\mathsf{c}})$ of strategy St is *accepting* if each downward path starting from a triple of the form $(0, q_0, in)$ is accepting. The following holds.

Proposition 5. *For each $w \in \Sigma^\omega$, $w \in \mathcal{L}(\mathcal{A})$* iff *$\mathcal{A}$ has a strategy* St *over w and an accepting path summary of* St.

Let $\Sigma_{ext} = \Sigma \times (2^{Q\times DIR\times Q\times Q}) \times (2^{Q\times[n+1]\times Q})^5$, where the partition in calls, returns, and internal actions is induced by Σ. For $w \in \Sigma^\omega$, $\mathsf{St} \in (2^{Q\times DIR\times Q\times Q})^\omega$, and a path summary $(\Delta, \Delta_\mathsf{a}, \Delta_\mathsf{c})$ of $\mathcal{A}$, the infinite word over Σ_{ext} associated with w, St, and $(\Delta, \Delta_\mathsf{a}, \Delta_\mathsf{c})$, written $(w, \mathsf{St}, (\Delta, \Delta_\mathsf{a}, \Delta_\mathsf{c}))$, is defined in the obvious way.

Theorem 3. *Given a parity* 2-AJA *$\mathcal{A}$, one can construct in* singly exponential *time a Büchi* NVPA *$\mathcal{P}_\mathcal{A}$ with size exponential in the size of $\mathcal{A}$ such that $\mathcal{L}(\mathcal{P}_\mathcal{A}) = \mathcal{L}(\mathcal{A})$.*

Proof. We first build a Büchi *NVPA* $\mathcal{P}^{ext}_\mathcal{A}$ over Σ_{ext} that accepts $(w, \mathsf{St}, (\Delta, \Delta_\mathsf{a}, \Delta_\mathsf{c}))$ iff St is a strategy of $\mathcal{A}$ over w and $(\Delta, \Delta_\mathsf{a}, \Delta_\mathsf{c})$ is an *accepting* path summary of St. The desired automaton $\mathcal{P}_\mathcal{A}$ is obtained by projecting out the St and $(\Delta, \Delta_\mathsf{a}, \Delta_\mathsf{c})$ components of the input word. $\mathcal{P}^{ext}_\mathcal{A}$ is the intersection of two Büchi *NVPA* $\mathcal{P}^{ext}_1$ and $\mathcal{P}^{ext}_2$.

$\mathcal{P}^{ext}_1$ checks that St is a strategy of $\mathcal{A}$ over w and $(\Delta, \Delta_\mathsf{a}, \Delta_\mathsf{c})$ is a path summary of St. These checks can be easily done as follows. $\mathcal{P}^{ext}_1$ keeps track by its finite control of the caller (if any) of the current input symbol and of the previous input symbol. On reading a call c_{ext}, $\mathcal{P}^{ext}_1$ pushes onto the stack the current caller and the call c_{ext}, and moves to the next input symbol updating the caller information to c_{ext}. Thus, in case the call c_{ext} returns (and the stack is popped), $\mathcal{P}^{ext}_1$ can check that the constraints between c_{ext} and the matching return are satisfied. Moreover, on reading c_{ext}, $\mathcal{P}^{ext}_1$ *either* (i) guesses that the matching return of c_{ext} does *not* exist and pushes onto the stack *also* the symbol 'NO' (the guess is correct iff 'NO' will be never popped from the stack), *or* (ii) guesses that the matching return of c_{ext} exists, pushes onto the stack also the symbol 'YES' and moves to a non-accepting state (the guess is correct iff 'YES' is eventually popped). In the second case, before 'YES' is eventually popped, whenever a call c_{new} occurs, the behaviour of $\mathcal{P}^{ext}_1$ is deterministic since the matching return of c_{new} must exists if the guess was correct (in this phase the symbols 'NO' and 'YES' are not used).

$\mathcal{P}^{ext}_2$ checks that $(\Delta, \Delta_\mathsf{a}, \Delta_\mathsf{c})$ is accepting. First we build a parity *NVPA* $\mathcal{P}_C$ that accepts $(w, \mathsf{St}, (\Delta, \Delta_\mathsf{a}, \Delta_\mathsf{c}))$ iff there is a downward path of St and $(\Delta, \Delta_\mathsf{a}, \Delta_\mathsf{c})$ that is *not* accepting. Essentially, $\mathcal{P}_C$ guesses such a path ρ and checks that it is not accepting. The computation of $\mathcal{P}_C$ is subdivided in phases. At the beginning of any phase, $\mathcal{P}_C$ keeps track by its finite control of the projection (q, in) of the current triple of the simulated path ρ, and chooses to start the simulation of a forward move, or of a zig-zag move, or of a terminal move. We describe the simulation of a zig-zag move (the other cases are simpler) with source (q, in, i) where i is a call position. Then, $\mathcal{P}_C$ guesses two triples $(q, in', q_r), (q^2_r, in'', q^2_c) \in Q \times [n] \times Q$. The first one must be in $\Delta_\mathsf{a}(h_r, \mathsf{down})$ where h_r is the matching return position of i, and the second one must be in $\Delta_\mathsf{a}(h^2_r, \mathsf{up})$, where $h^2_r < h_r$ and h^2_r is the matching return of a call representing the target of the zig-zag move. To check this $\mathcal{P}_C$ pushes onto the stack the triple (q, in', q_r) (when (q, in', q_r) is popped, $\mathcal{P}_C$ can check that the constraint on it is satisfied) and moves to the next input symbol in state (q^2_r, in'', q^2_c). $\mathcal{P}_C$ must also check that there is a finite path using only backward moves from the return position h_r in state q_r to the return h^2_r in state q^2_r. This part is discussed below. If the simulated move is the first zig-zag move along ρ, $\mathcal{P}_C$ pushes onto the stack also a special symbol '*'. Note that after the first zig-zag move, for each call visited by the remaining portion of ρ, the matching return exists. Thus, $\mathcal{P}_C$

must check the existence of the matching return only for the call associated with the first zig-zag move (this reduces to check that '*' is eventually popped).

$\mathcal{P}_C$ will remain in state (q_r^2, in'', q_c^2) (pushing some special symbol onto the stack whenever a call occurs) until[2] on reading a call c_{ext}, $\mathcal{P}_C$ guesses that c_{ext} is the matching call of position h_r^2. Thus, $\mathcal{P}_C$ pushes onto the stack the tuple $((q_r^2, in'', q_c^2), START)$ (recall that from this point, in state q_c^2, the path ρ can visit only positions in $[i_2, h_r^2[$, where i_2 is the position of c_{ext}). When the tuple is popped, $\mathcal{P}_C$ checks that the constraint on (q_r^2, in'', q_c^2) is satisfied and starts to simulate in reversed order backward moves starting from state q_r^2, until (on reading a return) a triple (q, in', q_r) is popped from the stack. The zig-zag move have been correctly simulated iff the current state of the guessed backward path is exactly q_r. The size of $\mathcal{P}_C$ is quadratic in the number of states of $\mathcal{A}$. The Büchi *NVPA* $\mathcal{P}_2^{ext}$ is obtained by complementating $\mathcal{P}_C$. By [4] the size of $\mathcal{P}_2^{ext}$ is exponential in the number of states of $\mathcal{A}$. This concludes the proof. □

***From Büchi* NVPA *to parity* 2-AJA.** For the converse translation, we can show that Büchi *1-AJA* are sufficient to capture the class of *VPL*.

Theorem 4. *Given a Büchi* NVPA $\mathcal{P}$*, we can build in polynomial time an equivalent Büchi* 1-AJA $\mathcal{A}_{\mathcal{P}}$ *whose number of states is quadratic in the number of states of* $\mathcal{P}$.

4.2 Decision Problems for Alternating Jump Automata

First, we consider emptiness and universality of (parity) *1-AJA* and *2-AJA*. Since the dual automaton of a *1-AJA* (resp., *2-AJA*) $\mathcal{A}$ has size linear in the size of $\mathcal{A}$ and accepts the complement of $\mathcal{L}(\mathcal{A})$, emptiness and universality of *1-AJA* and *2-AJA* are equivalent problems from the complexity point of view. For Büchi *NVPA*, emptiness is in PTIME and universality is EXPTIME-complete [4]. Thus, by Theorems 3 and 4, it follows that emptiness and universality of *1-AJA* and *2-AJA* are EXPTIME-complete.

For the pushdown model checking problem, given a pushdown system M and a *1-AJA* (resp., *2-AJA*) $\mathcal{A}$, checking whether $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{A})$ reduces to checking emptiness of $\mathcal{L}(M) \cap \mathcal{L}(\widetilde{\mathcal{P}}_{\mathcal{A}})$, where $\widetilde{\mathcal{P}}_{\mathcal{A}}$ is the *NVPA* equivalent to the dual of $\mathcal{A}$ (Theorem 3). This check can be done in time polynomial in the size of M and exponential in the size of $\mathcal{A}$. The problem is at least as hard as universality of *1-AJA* (resp., *2-AJA*), hence:

Theorem 5. *Emptiness and universality of* 1-AJA *and* 2-AJA *are* EXPTIME*-complete. Moreover, the pushdown model checking problem against* 1-AJA *(resp.,* 2-AJA*) specifications is* EXPTIME*-complete (and polynomial in the size of the pushdown system).*

5 Visibly Pushdown Linear-Time μ-Calculus (*VP-μTL*)

In this section we consider a linear-time fixpoint calculus, called *VP-μTL*, which subsumes CARET [3] and captures exactly the class of visibly pushdown languages.

VP-μTL extends the full linear-time μ-calculus (with both forward and backward modalities) introduced in [14] by allowing non-local forward and backward modalities:

[2] If, in the meanwhile, a triple (q, in', q_r) is popped from the stack, $\mathcal{P}_C$ rejects the input.

the *abstract next modality* allows to associate each call with its matching return (if any), and the (backward) *caller modality* allows to associate each position with its caller (if any). Thus, *VP-μTL* formulas are built from atomic actions over a pushdown alphabet Σ using boolean connectives, the standard next temporal operator $\bigcirc$ (here, denoted by $\bigcirc^{\downarrow}$), the standard backward version $\bigcirc^{\uparrow}$ of $\bigcirc$, the abstract version $\bigcirc^{\downarrow_{\mathsf{a}}}$ of $\bigcirc$, the caller version $\bigcirc^{\uparrow^{\mathsf{c}}}$ of $\bigcirc$, as well as the least (μ) and greatest (ν) fixpoint operators.

W.l.o.g. we assume that *VP-μTL* formulas are written in positive normal form (negation only applied to atomic actions). Let $Var = \{X, Y, \ldots\}$ be a finite set of variables and Σ be a pushdown alphabet. *VP-μTL* formulas φ over Σ and Var are defined as:

$$\varphi ::= a \mid \neg a \mid X \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc^{dir}\varphi \mid \neg\bigcirc^{dir}\,\mathtt{true} \mid \mu X.\varphi \mid \nu X.\varphi$$

where $a \in \Sigma$, $X \in Var$, and $dir \in \{\downarrow, \downarrow_{\mathsf{a}}, \uparrow, \uparrow^{\mathsf{c}}\}$. *VP-$\mu$TL* formulas φ are interpreted over words $w \in \Sigma^{\omega}$. Given a *valuation* $\mathcal{V} : Var \to 2^{\mathbb{N}}$ assigning a subset of $\mathbb{N}$ to each variable, the set of positions along w *satisfying* φ *under valuation* $\mathcal{V}$, written $\|\varphi\|_{\mathcal{V}}^{w}$, is defined as (we omit the clauses for atoms and boolean operators, which are standard):

$$\begin{array}{lcl}
\|X\|_{\mathcal{V}}^{w} & = & \mathcal{V}(X) \\
\|\bigcirc^{dir}\varphi\|_{\mathcal{V}}^{w} & = & \{i \in \mathbb{N} \mid succ(dir, w, i) \neq \top \text{ and } succ(dir, w, i) \in \|\varphi\|_{\mathcal{V}}^{w}\} \\
\|\neg\bigcirc^{dir}\,\mathtt{true}\|_{\mathcal{V}}^{w} & = & \{i \in \mathbb{N} \mid succ(dir, w, i) = \top\} \\
\|\mu X.\varphi\|_{\mathcal{V}}^{w} & = & \bigcap\{M \subseteq \mathbb{N} \mid \|\varphi\|_{\mathcal{V}[X \mapsto M]}^{w} \subseteq M\} \\
\|\nu X.\varphi\|_{\mathcal{V}}^{w} & = & \bigcup\{M \subseteq \mathbb{N} \mid \|\varphi\|_{\mathcal{V}[X \mapsto M]}^{w} \supseteq M\}
\end{array}$$

where $\mathcal{V}[X \mapsto M]$ maps X to M and behaves like $\mathcal{V}$ on the other variables. If φ does not contain free variables (φ is a *sentence*), $\|\varphi\|_{\mathcal{V}}^{w}$ does not depend on the valuation $\mathcal{V}$, and we write $\|\varphi\|^{w}$. The set of models of a sentence φ is $\mathcal{L}(\varphi) = \{w \in \Sigma^{\omega} \mid 0 \in \|\varphi\|^{w}\}$.

We can show that parity *2-AJA* are exactly as expressive as *VP-μTL* sentences.

Theorem 6. *Given a* VP-μTL *sentence φ, one can construct in* linear *time a parity* 2-AJA $\mathcal{A}_{\varphi}$ *such that* $\mathcal{L}(\mathcal{A}_{\varphi}) = \mathcal{L}(\varphi)$*. Vice versa, give a parity* 2-AJA $\mathcal{A}$*, one can construct in* linear *time a* VP-μTL *sentence* $\varphi_{\mathcal{A}}$ *such that* $\mathcal{L}(\varphi_{\mathcal{A}}) = \mathcal{L}(\mathcal{A})$*.*

By Theorems 6 and 5, we obtain the following result.

Corollary 1. *The satisfiability problem of* VP-μTL *is* EXPTIME*-complete. Moreover, the pushdown model checking problem against* VP-μTL *is* EXPTIME*-complete (and polynomial in the size of the pushdown system).*

6 Conclusion

We have investigated various classes of alternating automata over infinite structured words which capture exactly the class of visibly pushdown languages (*VPL*) [4]. First, we have shown that basic decision problems for alternating visibly pushdown automata (*AVPA*) are 2EXPTIME-complete. Second, we have introduced a new class of alternating finite-state automata, namely the class of *2-AJA*, and we have shown that basic decision problems for *2-AJA* are EXPTIME-complete. Finally, *2-AJA* have been used to obtain exponential-time completeness for satisfiability and pushdown model checking of a linear-time fixpoint calculus with past modalities, called *VP-μTL*, which subsumes the linear-time μ-calculus and the logic CARET [3].

We believe that *2-AJA* represent an elegant and interesting formulation of the theory of *VPL* on infinite words. In particular, boolean operations are easy and basic decision problems are equivalent from the complexity point of view and EXPTIME-complete. Moreover, *2-AJA* represent an intuitive extension of standard alternating finite-state automata on infinite words. Finally, and more importantly, *2-AJA* make easy the temporal reasoning about the past, and linear-time context-free temporal logics with past modalities such as CARET and the fixpoint calculus *VP-μTL* can be easily and linearly translated into *2-AJA*. Note that we have not considered (one-way or two-way) *nondeterministic* jump automata, since we can show that they capture only a proper subclass of the class of *VPL* (we defer further details to the full version of this paper).

References

1. Alur, R., Chaudhuri, S., Madhusudan, P.: A fixpoint calculus for local and global program flows. In: Proc. 33rd POPL, pp. 153–165. ACM Press, New York (2006)
2. Alur, R., Chaudhuri, S., Madhusudan, P.: Languages of nested trees. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 329–342. Springer, Heidelberg (2006)
3. Alur, R., Etessami, K., Madhusudan, P.: A Temporal Logic of Nested Calls and Returns. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 467–481. Springer, Heidelberg (2004)
4. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Proc. 36th STOC, pp. 202–211. ACM Press, New York (2004)
5. Alur, R., Madhusudan, P.: Adding nesting structure to words. In: Developments in Language Theory, pp. 1–13 (2006)
6. Ball, T., Rajamani, S.: Bebop: a symbolic model checker for boolean programs. In: Havelund, K., Penix, J., Visser, W. (eds.) SPIN Model Checking and Software Verification. LNCS, vol. 1885, pp. 113–130. Springer, Heidelberg (2000)
7. Bouajjani, A., Esparza, J., Maler, O.: Reachability Analysis of Pushdown Automata: Application to Model-Checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997)
8. Chatterjee, K., Ma, D., Majumdar, R., Zhao, T., Henzinger, T.A., Palsberg, J.: Stack size analysis for interrupt-driven programs. In: Cousot, R. (ed.) SAS 2003. LNCS, vol. 2694, pp. 109–126. Springer, Heidelberg (2003)
9. Chen, H., Wagner, D.: Mops: an infrastructure for examining security properties of software. In: Proc. 9th CCS, pp. 235–244. ACM Press, New York (2002)
10. Esparza, J., Kucera, A., Schwoon, S.: Model checking LTL with regular valuations for pushdown systems. Information and Computation 186(2), 355–376 (2003)
11. Loding, C., Madhusudan, P., Serre, O.: Visibly pushdown games. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 408–420. Springer, Heidelberg (2004)
12. Muller, D.E., Schupp, P.E.: Alternating Automata on Infinite Trees. Theoretical Computer Science 54, 267–276 (1987)

13. Muller, D.E., Schupp, P.E.: Simulating alternating tree automata by nondeterministic automata: new results and new proofs of the Theorems of Rabin, McNaughton and Safra. Theoretical Computer Science 141(1-2), 69–107 (1995)
14. Vardi, M.Y.: A temporal fixpoint calculus. In: Proc. 15th Annual POPL, pp. 250–259. ACM Press, New York (1988)
15. Vardi, M.Y.: Reasoning about the past with two-way automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 628–641. Springer, Heidelberg (1998)
16. 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)

Temporal Antecedent Failure: Refining Vacuity

Shoham Ben-David[1], Dana Fisman[2,3], and Sitvanit Ruah[3]

[1] David R. Cheriton School of Computer Science University of Waterloo
[2] School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel
[3] IBM Haifa Research Lab, Haifa University Campus, Haifa 31905, Israel

Abstract. We re-examine vacuity in temporal logic model checking. We note two disturbing phenomena in recent results in this area. The first indicates that not all vacuities detected in practical applications are considered a problem by the system verifier. The second shows that vacuity detection for certain logics can be very complex and time consuming. This brings vacuity detection into an undesirable situation where the user of the model checking tool may find herself waiting a long time for results that are of no interest for her.

In this paper we define *Temporal Antecedent Failure*, an extension of antecedent failure to temporal logic, which refines the notion of vacuity. According to our experience, this type of vacuity *always* indicates a problem in the model, environment or formula. On top, detection of this vacuity is extremely easy to achieve. We base our definition and algorithm on regular expressions, that have become the major temporal logic specification in practical applications.

Keywords: Model checking, Temporal logic, Vacuity, Regular expressions, PSL, SVA, Antecedent failure.

1 Introduction

Model checking ([15,28], c.f.[16]) is the procedure of deciding whether a given model satisfies a given formula. One of the nice features of model checking is the ability to produce, when the formula does not hold in the model, an execution path demonstrating the failure. However, when the formula is found to hold in the model, traditional model checking tools provide no further information. While satisfaction of the formula in the model should usually be considered "good news", it is many times the case that the positive answer was caused by some error in the formula, model or environment. As a simple example consider the LTL formula $\varphi = \mathsf{G}\,(req \rightarrow \mathsf{F}\,(ack))$, stating that every request req must eventually be acknowledged by ack. This formula holds in a model in which req is never active. In this case we say that the formula is *vacuously* satisfied in the model.

Vacuity in temporal logic model checking was introduced by Beer et al ([3,4]), who defined it as follows: a formula φ is vacuously satisfied in a model M iff it is satisfied in M, (i.e., $M \models \varphi$) and there exists a subformula ψ of φ that *does not affect* the truth value of φ in M. That is, there exists a subformula ψ of φ such that $M \models \varphi[\psi \leftarrow \psi']$ for any formula ψ'.[1] For example, in the formula $\varphi = \mathsf{G}\,(req \rightarrow \mathsf{F}\,(ack))$, if req

[1] We use $\varphi[\psi \leftarrow \psi']$ to denote the formula obtained from φ by replacing the subformula ψ of φ with ψ'.

is never active in M, then the formula holds, no matter what the value of $\mathsf{F}\,(ack)$ is. Thus, the subformula $\mathsf{F}\,(ack)$ does not affect the truth value of φ in M. Note that this definition ignores the question of the *reason* for a vacuous satisfaction. In our example the reason for vacuity is the failure of req to ever hold in the model.

Since the introduction of vacuity, research in this area concentrated mainly on extending the languages and methods to which vacuity could be applied [24,27,17,26,20,30]. Armoni et al. in [1] defined a new type of vacuity, which they called *trace vacuity*. Chechik and Gurfinkel in [19] showed that detecting trace vacuity is exponential for CTL, and double-exponential for CTL*. Bustan et al. in [13] showed that detecting trace vacuity for RELTL (an extension of LTL with a regular layer), involves an exponential blow-up in addition to the standard exponential blow-up for LTL model checking. They suggested that weaker vacuity definition should be adopted for practical applications.

In a recent paper [14] Chechik et al. present an interesting observation. They note that in many cases, vacuities detected according to existing definitions are not considered a problem by the verifier of the system. For example, consider again the formula $\varphi = \mathsf{G}\,(req \rightarrow \mathsf{F}\,(ack))$, and assume the system is designed in such a way that ack is given whenever the system is ready. Thus, if the system behaves correctly, it infinitely often gets to a "ready" state, and the formula $\mathsf{F}\,(ack)$ always holds. According to the definition of vacuity, φ holds vacuously in the model (since req does not affect the truth value of φ in the model), although no real problem exists. This phenomenon, coupled with the high complexity results reported for some logics, brings vacuity detection into an undesirable situation where the user of the model checking tool may find herself waiting a long time for results that are of no interest for her.

In this paper we approach the problem by providing a refined definition of vacuity, such that detection is almost "for free", and, according to feedback from users, vacuity *always* indicate real problems. We define *temporal antecedent failure* (TAF), a type of vacuity that occurs when some *pre-condition* in the formula is never fulfilled in the model. The definition of TAF is semantic and is therefore unlikely to suffer from typical problems with syntactic definitions as shown in Armoni et al. [1].

We work with formulas given in terms of regular expressions, that are the major specification method used in practice [8,18,23,22] . We show how vacuity can be detected by asserting an *invariant* condition on the automaton already built for the original formula. We further use these invariant conditions to determine the *reason* for vacuity, when detected. Vacuity and its reasons are therefore detected with almost no overhead on the state space.

Note that the notion of *vacuity reasons*, formally defined in section 3.3, is different from *vacuity causes* defined by Samer and Veith in [29]. Samer and Veith introduce *weak vacuity* which occurs when a sub-formula can be replaced by a stronger sub-formula without changing the truth value (where a formula φ is stronger than ψ if φ implies ψ). Since the set of candidates for stronger formulas is large and not all stronger formulas potentially provide a trivial reason for satisfaction of the original formula they ask the user to provide a set of stronger formulas which they term *causes* according to which *weak vacuity* will be sought for. In contrast, in our work a *reason* is a Boolean expression appearing at the original formula which is responsible for the detected vacuity.

We note that our approach resembles that of the original vacuity work of Beer et al. [3], who considered *antecedent failure* [2] as their motivation. While our definition of TAF generalizes antecedent failure in the temporal direction, their definition of vacuity generalizes antecedent failure even in propositional logic — where the dimension of time is absent. We further note that the algorithm provided in [3] concentrated mainly on TAF. Thus our experience of vacuities being 100% interesting, comply with their report of vacuities "... always point to a real problem in either the design or its specification or environment" [3].

The rest of the paper is organized as follows. The next section gives preliminaries. Section 3 defines temporal antecedent failure and its reasons, and shows that formulas that may suffer from TAF can be represented by regular expressions. Section 4 gives an efficient algorithm to detect TAF and its reasons. Section 5 concludes the paper and discusses related work. Due to lack of space, all proofs are omitted. They can be found in the full version of the paper [9].

2 Preliminaries

2.1 Notations

We denote a letter from a given alphabet Σ by ℓ, and an empty, finite, or infinite word from Σ by u, v, or w (possibly with subscripts). The *concatenation* of u and v is denoted by uv. If u is infinite, then $uv = u$. The empty word is denoted by ϵ, so that $w\epsilon = \epsilon w = w$. If $w = uv$ we say that u is a *prefix* of w, denoted $u \preceq w$, that v is a *suffix* of w, and that w is an *extension* of u, denoted $w \succeq u$.

We denote the *length* of a word v by $|v|$. The empty word ϵ has length 0, a finite word $v = (\ell_0\ell_1\ell_2 \cdots \ell_n)$ has length $n + 1$, and an infinite word has length ∞. Let i denote a non-negative integer. For $i < |v|$ we use v^i to denote the $(i + 1)^{th}$ letter of v (since counting of letters starts at zero), and v^{-i} to denote the $(i + 1)^{th}$ letter of v counting backwards. That is if $v = (\ell_n \ldots \ell_2\ell_1\ell_0)$ then $v^{-i} = \ell^i$. Let j and k denote integers such that $j \leq k$. We denote by $v^{j..}$ the suffix of v starting at v^j and by $v^{j..k}$ the subword of v starting at v^j and ending at v^k.

We denote a set of finite/infinite words by U, V or W and refer to them as *properties*. The *concatenation* of U and V, denoted UV, is the set $\{uv \mid u \in U,\ v \in V\}$. Define $V^0 = \{\epsilon\}$ and $V^k = VV^{k-1}$ for $k \geq 1$. The $*$-*closure* of V, denoted V^*, is the set $V^* = \bigcup_{k<\omega} V^k$. The notation V^+ is used for the set $\bigcup_{0<k<\omega} V^k$. The infinite concatenation of V to itself is denoted V^ω.

In this work we follow the linear time framework. Thus, we assume the underlying temporal logic is linear and we regard a model M as an ω-regular property (that is, we regard M as a set of infinite words that is generated by an automaton).

2.2 Temporal Logic

Since the underlying temporal logic is linear, the models satisfying a temporal formula are words (finite or infinite). A temporal logic formula may refer to the past, to the future or to both. We use the term *past* formulas for formulas that do not refer to the strict future. We use the term *future* formulas for formulas that do not refer to the strict past. We use the term *present* formula for formulas that do not refer to the strict future or to the strict past. The set of models satisfying a past formula is composed of finite non-empty words. The set of models satisfying future formulas is composed of infinite words. The set of models satisfying a present formula is composed of single letters. The set of models satisfying an arbitrary formula is composed of pairs $\langle w, i\rangle$ where w is an infinite word and i is a non-negative integer. The prefix $w^{0..i}$ represents the past and present and the suffix $w^{i..}$ represents the present and future.

Below we give the semantics of the commonly used operators: X (next), $\overleftarrow{\mathsf{X}}$ (previous), G (globally), $\overleftarrow{\mathsf{G}}$ (historically), F (eventually), $\overleftarrow{\mathsf{F}}$ (once), U (until), and $\overleftarrow{\mathsf{U}}$ (since). Let $\mathbb{P}$ be a given set of atomic propositions. We identify the set of present formulas $\mathbb{B}$ with the set $2^{2^{\mathbb{P}}}$ of subsets of valuation to the atomic propositions $\mathbb{P}$. We use *true* and *false* to denote the elements $2^{\mathbb{P}}$ and $\emptyset$ of $\mathbb{B}$, respectively. Let b be a present formula and φ an arbitrary formula.

- $\langle w, i\rangle \models b \iff w^i \in b.$
- $\langle w, i\rangle \models \mathsf{X}\,\varphi \iff \langle w, i+1\rangle \models \varphi.$
- $\langle w, i\rangle \models \overleftarrow{\mathsf{X}}\,\varphi \iff i > 0$ and $\langle w, i-1\rangle \models \varphi.$
- $\langle w, i\rangle \models \mathsf{G}\,\varphi \iff \forall j \geq i,\ \langle w, j\rangle \models \varphi.$
- $\langle w, i\rangle \models \overleftarrow{\mathsf{G}}\,\varphi \iff \forall j \leq i,\ \langle w, j\rangle \models \varphi.$
- $\langle w, i\rangle \models \mathsf{F}\,\varphi \iff \exists j \geq i,\ \langle w, j\rangle \models \varphi.$
- $\langle w, i\rangle \models \overleftarrow{\mathsf{F}}\,\varphi \iff \exists j \leq i,\ \langle w, j\rangle \models \varphi.$
- $\langle w, i\rangle \models \varphi \mathsf{U}\, \psi \iff \exists k \geq i,\ \langle w, k\rangle \models \psi$ and $\forall i \leq j < k,\ \langle w, j\rangle \models \varphi.$
- $\langle w, i\rangle \models \varphi \overleftarrow{\mathsf{U}}\, \psi \iff \exists k \leq i,\ \langle w, k\rangle \models \psi$ and $\forall i \geq j > k,\ \langle w, j\rangle \models \varphi.$

A formula φ is *initially true* on a word w iff $w, 0 \models \varphi$. Given a temporal logic formula φ we use $[\![\varphi\rangle\!\rangle$ to denote the set of words initially satisfying φ. That is, $[\![\varphi\rangle\!\rangle = \{w \ :\ \langle w, 0\rangle \models \varphi\}$. For a past formula φ, we use $\langle\!\langle\varphi]\!]$ to denote the set of finite words satisfying φ at their last letter. That is, $\langle\!\langle\varphi]\!] = \{w \ :\ w$ is finite and $\langle w, |w|-1\rangle \models \varphi\}$.

2.3 Regular Expressions

Definition 1 (Regular Expressions (RE s))

- *Let* $\mathbb{B}$ *be a finite non-empty set, and let* b *be an element in* $\mathbb{B}$*. The set of* regular expressions (REs) over $\mathbb{B}$ is recursively defined as follows: $r\ ,= b \mid r{\cdot}r \mid r \cup r \mid r^*$
- The *language of a regular expression over* $\mathbb{B}$ *with respect to* $\mathfrak{L}(\cdot)$ *is recursively defined as follows, where* $b \in \mathbb{B}$*, and* r_1 *and* r_2 *are* RE*s.*

1. $[\![b]\!] = \mathfrak{L}(b)$	*3.* $[\![r_1 \cup r_2]\!] = [\![r_1]\!] \cup [\![r_2]\!]$
2. $[\![r_1{\cdot}r_2]\!] = [\![r_1]\!][\![r_2]\!]$	*4.* $[\![r^*]\!] = [\![r]\!]^*$

In standard definition of regular expressions $\mathfrak{L}(b) = \{b\}$ in the context of temporal logic $\mathbb{B}$ is a set of present formulas over a given set of atomic propositions $\mathbb{P}$ and $\mathfrak{L}(b) \subseteq 2^{2^{\mathbb{P}}}$ is the set of assignments to the propositions $\mathbb{P}$ for which b holds.

2.4 Positions in Regular Expressions and Related Partial Orders

An RE is called *linear* if no letter appears in it more than once [10]. Any RE can be linearized by subscripting its letters with unique indexes.

Example 1. Let r_1 be the RE $\{a{\cdot}\{\{b^*{\cdot}c\}{\cup}\{d{\cdot}e^*\}\}{\cdot}c{\cdot}e\}$. It can be subscriptized to the linear RE $r_1' = \{a_1{\cdot}\{\{b_2^*{\cdot}c_3\}{\cup}\{d_4{\cdot}e_5^*\}\}{\cdot}c_6{\cdot}e_7\}$.

We call the letters of the subscriptized RE *position*s and save the term *letter*s for the deindexed positions. That is, the positions of r_1' are a_1, b_2, c_3, d_4, e_5, c_6 and e_7 and its letters are a, b, c, d and e — the same as the letters of r_1. The set of positions of an RE r is denoted $pos(r)$. We can define a partial order between positions of a given RE. The definition of the partial order makes use of the following functions:

Definition 2 (Position Functions). *Let r be a linear* RE*, and let $[\![r]\!]$ be defined as in Definition 1 where for a position b, $\mathfrak{L}(b)$ is $\{b\}$.*

– $\mathcal{F}(r)$ *- the set of positions that match the* first *letter of some word in* $[\![r]\!]$.
  Formally, $\mathcal{F}(r) = \{x \in pos(r) \mid \exists v \in pos(r)^* \text{ s.t. } xv \in [\![r]\!]\}$.
– $\mathcal{L}(r)$ *- the set of positions that match the* last *letter of some word in* $[\![r]\!]$.
  Formally, $\mathcal{L}(r) = \{x \in pos(r) \mid \exists v \in pos(r)^* \text{ s.t. } vx \in [\![r]\!]\}$.
– $\mathcal{N}(r,x)$ *- the set of positions that can* follow *position x in a path through r.*
  Formally, $\mathcal{N}(r,x) = \{y \in pos(r) \mid \exists u,v \in pos(r)^* \text{ s.t. } uxyv \in [\![r]\!]\}$.
– $\mathcal{P}(r,x)$ *- the set of positions that can* precede *position x in a path through r.*
  Formally, $\mathcal{P}(r,x) = \{y \in pos(r) \mid \exists u,v \in pos(r)^* \text{ s.t. } uyxv \in [\![r]\!]\}$.

An inductive definition of these functions appears in [6] and is repeated in the full version of the paper [9].

The transitive closure of $\mathcal{N}(\cdot)$ induces a partial order $\prec\!\!\cdot$ on the set of positions so that, $\mathcal{F}(r)$ positions are the minimal positions, $\mathcal{L}(r)$ are the maximal positions, if $y \in \mathcal{N}(r,x)$ then $y \cdot\!\!\succ x$, and if $z \cdot\!\!\succ y$ and $y \cdot\!\!\succ x$ then $z \cdot\!\!\succ x$.

Example 2. For r_1' defined in Example 1 we have $\mathcal{F}(r_1') = \{a_1\}$, $\mathcal{L}(r_1') = \{e_7\}$. The parital order is the transitive closure of the relation given by: $a_1 \prec\!\!\cdot\, b_2$, $b_2 \prec\!\!\cdot\, c_3$, $c_3 \prec\!\!\cdot\, c_6$, $c_6 \prec\!\!\cdot\, e_7$, $a_1 \prec\!\!\cdot\, d_4$, $d_4 \prec\!\!\cdot\, e_5$ and $e_5 \prec\!\!\cdot\, c_6$.

Given an RE r and a position x we denote by $r_{x\rfloor}$ the "prefix" of r that ends with x. Similarly, we denote by $r\lfloor_x$ the "prefix" of r that ends exactly before x. Formally,

1. $b_{x\rfloor} = b$
2. $r_1{\cdot}r_{2\,x\rfloor} = \begin{cases} r_{1\,x\rfloor} & \text{if } x \in r_1 \\ r_1{\cdot}r_{2\,x\rfloor} & \text{otherwise} \end{cases}$
3. $r_1{\cup}r_{2\,x\rfloor} = \begin{cases} r_{1\,x\rfloor} & \text{if } x \in r_1 \\ r_{2\,x\rfloor} & \text{otherwise} \end{cases}$
4. $r^*{}_{x\rfloor} = r_{x\rfloor}$

1. $b\lfloor_x = \epsilon$
2. $r_1{\cdot}r_2\lfloor_x = \begin{cases} r_1\lfloor_x & \text{if } x \in r_1 \\ r_1{\cdot}r_2\lfloor_x & \text{otherwise} \end{cases}$
3. $r_1{\cup}r_2\lfloor_x = \begin{cases} r_1\lfloor_x & \text{if } x \in r_1 \\ r_2\lfloor_x & \text{otherwise} \end{cases}$
4. $r^*\lfloor_x = r\lfloor_x$

Example 3. For r_1' defined in Example 1, we have $r'_{1\,e_5\rfloor} = \{a_1 \cdot d_4 \cdot e_5\}$, $r_1'\lfloor_{e_5} = \{a_1 \cdot d_4\}$, $r'_{1\,c_6\rfloor} = \{a_1{\cdot}\{b_2^*{\cdot}c_3\}{\cup}\{d_4{\cdot}e_5^*\}{\cdot}c_6\}$ and $r_1'\lfloor_{c_6} = \{a_1{\cdot}\{b_2^*{\cdot}c_3\}{\cup}\{d_4{\cdot}e_5^*\}\}$.

3 Temporal Antecedent Failure (TAF) and Its Reasons

Antecedent failure in propositional logic [2] occurs when the formula is trivially valid because the pre-condition (antecedent) of the formula is not satisfiable in the model. We generalize this notion to *Temporal Antecedent Failure* (TAF), where some future requirement depends on a temporal pre-condition. We start by giving a definition of TAF in terms of past and future formulas. In section 3.2 we show that any formula that may suffer from TAF can be expressed using regular expressions. When a temporal pre-condition fails to hold, there could be different reasons for that. We define TAF reasons in section 3.3.

3.1 Temporal Antecedent Failure

In propositional logic a formula suffers from antecedent failure if it is of the form $(A \rightarrow B)$ and A is not satisfiable. Intuitively, the generalization to temporal logic is that a formula suffers from *temporal antecedent failure* if it is of the form $\mathsf{G}\,(\Phi \rightarrow \Psi)$ where Φ is a past formula representing a condition, Ψ is a future formula representing a requirement, and the condition Φ never holds. To capture this intuition, we present the following two definitions.

Definition 3 (Temporal Implication Form). *A formula* has a temporal implication form iff it can be expressed in the form $\mathsf{G}\,(\Phi \rightarrow \Psi)$ where Φ is a past formula and Ψ is a future formula.

We note that Ψ is not required to refer to the strict future, that is, it may also refer to the present. We also note that this allows Ψ to be the present formula *false* or another unsatisfiable formula. In such a case Ψ does not carry a requirement, but rather states that Φ is disallowed. Therefore, Φ is not really an antecedent and the notion of antecdent failure does not apply. We therefore assume Ψ is satisfiable.

Definition 4 (Temporal Antecedent Failure). *Let M be a model, Φ a past formula and Ψ a future formula. We say that the formula $\varphi = \mathsf{G}(\Phi \rightarrow \Psi)$* suffers from TAF of Φ in model M iff $M \models \varphi$ and forall words w in M, and forall $i < |w|$ we have $\langle w, i \rangle \not\models \Phi$.

We say that a *property* has a temporal implication form if it can be expressed by a formula that has a temporal implication form. We say that a *property* suffers from temporal antecedent failure in M if it can be expressed by a formula that suffers from temporal antecedent failure in M. In the sequel we use antecedent failure to mean temporal antecedent failure. Also, when M is understood from the context we sometimes omit it.

3.2 Moving to Regular Expressions

We examine formulas in temporal implication form in view of the IEEE standard temporal logic PSL [18,23]. A major feature of PSL is the *suffix implication* operator, which makes use of regular expressions. A suffix implication formula is of the form $r \mapsto \varphi$ where r is a regular expression and φ is a PSL formula. Intuitively, this formula states that whenever a prefix of a given path matches the regular expression r, the suffix of that path should satisfy the requirement φ.

Example 4. The following formula

$$\psi = \{\mathit{true}^* \cdot \mathit{req} \cdot \neg \mathit{ack}^* \cdot \mathit{ack} \cdot \neg \mathit{abort}\} \mapsto (\neg \mathit{retry}\ \mathsf{U}\ \mathit{req})$$

states that if the first ack following req is not $abort$ed one cycle after ack, then it must not be retried (signal $retry$ should not hold at least until the next req holds). The formal definition of suffix implication taken from [23], is given below.

Definition 5 (Suffix Implication Formulas [23]). *Let r be a regular expression and φ a temporal logic formula, both over a given set of atomic propositions $\mathbb{P}$. Let v be a word over $2^{\mathbb{P}}$. Then*

$$v \models r \mapsto \varphi \iff \forall j < |v|,\ \mathit{if}\ v^{0..j} \in [\![r]\!]\ \mathit{then}\ \langle v, j\rangle \models \varphi$$

The following claim connects formulas in temporal implication form to suffix implication formulas.

Claim 1. *An omega regular property has a temporal implication form iff it can be expressed in the form $r \mapsto \Psi$ where r is a regular expression and Ψ is a future formula.*

We note that although PSL includes other language constructs, suffix implication is the major one and in fact, other temporal formulas can be translated into suffix implication [8,11].[2] Note further that a property may be expressed by two different formulas in suffix implication form. For example, the formula $\{a \cdot b\} \mapsto (\mathsf{F}\, c)$ is equivalent to the formula $\{a\} \mapsto \mathsf{X}\,(\neg b \vee (b \wedge \mathsf{F}\, c))$. In order to get maximum information regarding TAF it is beneficial to take a form in which the left-hand-side of suffix implication is maximal in terms of the partial order between REs. Indeed, the procedure in [8] yields a formula in which the RE at the left-hand-side is maximal.[3] In the rest of the paper we shall assume the formula is given as a suffix implication.

The following claim provides a characterization of TAF in terms of suffix implication formulas.

Claim 2. *A formula of the form $r \mapsto \Psi$ where r is a regular expression and Ψ is a future formula suffers from* TAF *of r in model M iff forall words w in M, and forall prefixes $u \preceq w$ we have $u \notin [\![r]\!]$.*

Example 5. Consider the formula ψ given in Example 4, for which $r = \{\mathit{true}^* \cdot \mathit{req} \cdot \neg \mathit{ack}^* \cdot \mathit{ack} \cdot \neg \mathit{abort}\}$. The formula ψ would suffer from TAF in a model M where no word ever satisfies r. (That is, there is never a sequence of req, followed by an ack after

[2] As elaborated in [8], the majority of formulas written in practice can be effectively transformed into suffix implication form. In particular, the common fragment of LTL and ACTL, all RCTL formulas and all the safety formulas in the simple subset of PSL can be linearly translated into suffix implication form. Clearly, since the left hand side of the suffix implication operator can be any formula, we can manage in the same manner liveness formulas as well [11].

[3] The procedure in [8] yields a formula of the form $\{r\} \mapsto \mathit{false}$. As explained earlier, for TAF detection we assume the right-hand-side is not a contradiction. However, it is easy to transform the result of [8] to one in which the right-hand-side is not a contradiction.

a finite number of cycles, and then $\neg abort$ one cycle after ack.) Note that there could be several reasons for this r never to hold on any word. It could be that no reqs are ever given in M, or no acks are given. It could also be the case that in our model, all reqs are always $abort$ed. In the next section we define what it means to be a *reason* for TAF.

Note that Ψ plays no role in the decision of whether $r \mapsto \Psi$ suffers from TAF in a given model M. Thus, if $r \mapsto \Psi$ suffers from TAF of r in M then for any other formula Ψ', the formula $r \mapsto \Psi'$ also suffers from TAF of r in M. Therefore, from now on we say that r suffers from TAF in M without specifying the formula.

3.3 Defining TAF Reasons

Let r be an RE that suffers from TAF in a model M. We would like to find the reason for this. For example if $r = \{true^* \cdot a \cdot b \cdot c \cdot c^* \cdot d\}$ and b never holds after an a, we would like to say that b is the reason for TAF. Note that in addition, it might be the case that d never holds after c, but we still view b as the reason. That is, our intuitive view of the reason for TAF is the earliest letter that does not hold when expected. Having said that, we note that sometimes the earliest letter that does not hold when expected, is actually *not* the reason. To see why, consider the following example.

Example 6. Let $r_1' = \{a_1 \cdot \{b_2^* \cdot c_3\} \cup \{d_4 \cdot e_5^*\} \cdot c_6 \cdot e_7\}$ be the RE defined in Example 1 and assume its letters are mutually exclusive (that is, if one holds the others do not). Let M be a model such that a_1 holds at the first cycle, b_2 holds at the second cycle, c_3 at the third cycle and c_6 never holds. Thus, c_6 is a reason for TAF, although there exists an earlier position, d_4, that also does not hold when expected.

The formal definition of TAF reasons follows. Intuitively, a TAF reason is a position that does not hold when expected although one of its immediate predecessors does hold when expected.

Definition 6 (Temporal Antecedent Failure Reason). *Let M be a model and r a linear* RE*, such that r suffers from* TAF *in M. A* position x is a reason for TAF of r in M iff $r_{x\rfloor}$ suffers from TAF in M and there exists a position $x' \in \mathcal{P}(r, x)$ such that $r_{x'\rfloor}$ does not suffer from TAF in M.

The following proposition assures that if a model M suffers from TAF with respect to some RE r then indeed (at least) one of the positions in r is a reason for TAF of r in M.

Proposition 1. *Let M be a model and r an* RE*, such that r suffers from* TAF *in M. Then there exists a position x in r such that x is a reason for* TAF *of r in M.*

Consider again the RE r_1 and the model M of Example 6. We note that according to Definition 6 the position d_4 is also a reason for TAF. However, c_6 is a more fundamental reason: the fact that c_6 does not hold when expected implies that for any Ψ the formula $r \mapsto \Psi$ holds. The fact that d_4 does not hold when expected does not imply that, as it could be that b_2, c_3, c_6, e_7 do hold when expected and thus in the formula $r \mapsto \Psi$, the requirement Ψ must be fulfilled. We thus differentiate between two types of reasons for antecedent failure. If the failure of the position is "recovered" by another position, it is a *secondary* reason. Otherwise it is a *primary* reason.

Definition 7 (Primary and Secondary Reasons). *Let M be a model and r a linear* RE. *Given position x is a reason of antecedent failure of r in M, we say that x is a* primary reason if forall $x' \succ x$ we have that $r_{x'}\rfloor$ suffers from TAF in M and x' is not a reason for TAF of r in M. Otherwise we say that x is a *secondary reason.*

One can check that these definitions indeed give us that in Example 6 both d_4 and c_6 are reasons, and c_6 is a primary reason while d_4 is a secondary reason. There are no other reasons for TAF of r'_1 in this example. As another example consider the RE $r_2 = \{true^* \cdot a \cdot b^* \cdot c\}$ and assume the model is such that a holds only on the 11^{th} cycle and neither b nor c hold on the 12^{th} cycle. Then both b and c are reasons, and c is a primary reason while b is a secondary reason. There are no other reasons for TAF of r_2 in this example.

The following proposition strengthens Proposition 1 by asserting that if r suffers from TAF in M then there exists a *primary* reason for this.

Proposition 2. *Let M be a model and r an* RE*, such that r suffers from* TAF *in M. Then there exists a position x in r such that x is a primary reason for* TAF *of r in M.*

4 Implementation

4.1 Detecting Antecedent Failure and Its Reasons

Below we provide the implementation of detection of antecedent failure and its reasons, for suffix implication formulas. A key feature of our algorithm is that it does not add auxiliary automata, but rather adds invariant checks on the automata built for model checking the formula. Thus, before providing our algorithm we recall several facts regarding the model checking of suffix implication formulas, using the automata theoretic approach.

The first fact we build upon is that for any regular expression there exists a natural NFA accepting the same language. An NFA N is said to be *natural* for a linear RE r if every state of N (except possibly a trapping state) is associated with a unique position in r. A construction for a linear NFA for a given RE r can be found in [6] and is repeated in the full version of the paper [9]. This NFA also satisfies that all the outgoing edges from a given state have the same label, and the label is the set of valuations satisfying the letter associated with the state.

The following proposition [12, Proposition 3.5] states that this NFA can be used for both recognizing a given RE and failing to recognize it.

Proposition 3 ([12]). *Let r be an* RE. *There exists an* NFA *N_r with $O(|r|)$ states such that N_r has a trapping non-accepting state q_{bad} and for every word w over Σ,*

1. *there exists an accepting run of N_r on w iff there exists a non-empty prefix $u \preceq w$ such that $u \in [\![r]\!]$.*
2. *every run of N_r on w reaches q_{bad} iff for every non-empty prefix $u \preceq w$ we have $u \notin [\![r]\!]$.*

We refer to such an NFA as a *trapping* NFA. The proof of the following proposition from [12, Proposition 3.15] asserts that efficient model checking a formula of the form $r \mapsto \varphi$ can be done by "concatenating" the trapping NFA for r with the Büchi automaton for φ.

Proposition 4 ([12]). *Let r be an* RE*, and φ a formula. If there exists a weak (terminal) Büchi automaton for $\neg\varphi$ with state set S, then there exists a weak (terminal) Büchi automaton for $\neg(r \mapsto \varphi)$ with at most $O(|r| + |S|)$ states.*

We are now ready for the implementation. The following proposition states that antecedent failure of $r \mapsto \varphi$ with respect to r in M can be detected by checking the invariant property stating that the trapping NFA of r does not reach an accepting state on a parallel composition of the model with the trapping NFA of r. We represent the model M and the NFA for r symbolically using discrete transition systems (DTSs) over finite and infinite words. The definition of a DTS, the transition from an NFA to a DTS and their parallel composition, denoted $|||$ are defined in [7] and repeated in the full version of the paper [9]. In the sequel we use $\mathcal{D}_M$ for the DTS representation of M. We use $\mathcal{D}_r$ to denote the DTS representation of the trapping NFA for r and $\mathcal{A}_r$ to denote the set of accepting states of $\mathcal{D}_r$. For a position x of r we use at_x to denote a present formula stating that $\mathcal{D}_r$ is in the state corresponding to position x. We use $\ell(x)$ to denote the letter on the outgoing edges from state satisfying at_x. Thus $\ell(x)$ is the letter obtained from position x after removing the added subscript.

Proposition 5. *Let M be a model and r an* RE.

$$r \textit{ suffers from } \text{TAF} \textit{ in } M \textbf{\textit{ iff }} \mathcal{D}_M \,|||\, \mathcal{D}_r \models \mathsf{AG}\, \neg\mathcal{A}_r$$

Proposition 6 below states the invariants that provide a detection of a TAF reason. The first invariant checks that the position is reachable, implying that an immediate predecessor did hold when expected. The second invariant holds if the position does not hold when expected. Proposition 7 below provides the invariant that distinguishes between primary and secondary reasons: it checks whether there exists a (not necessarily immediate) successor that is reachable.

Proposition 6. *Let M be a model, r an* RE *such that M suffers from antecedent failure of r. Let x be a position in r. Then, x is a* reason of antecedent failure in φ **iff** the following two conditions hold:

– $\mathcal{D}_M \,|||\, \mathcal{D}_r \not\models \mathsf{AG}\ \neg at\_x$ // x is reachable.
– $\mathcal{D}_M \,|||\, \mathcal{D}_r \models \mathsf{AG}\ (at\_x \rightarrow \neg\ell(x))$ // x is not exercised.

Proposition 7. *Let M be a model, r an* RE *and x a position in r that is a reason of antecedent failure. Then, x is a primary reason of antecedent failure iff $\mathcal{D}_M \,|||\, \mathcal{D}_r \models \mathsf{AG} \bigwedge_{y \succ x} \neg at\_y$. Otherwise x is a secondary reason.*

It is important to note that all the detection "tests" are phrased as a set of invariants over the parallel execution of the given model with the NFA for the given formula. Thus, they can run on the fly over the same model (without adding any automata) and therefore the cost of adding them to the verification is small.

4.2 Witness Generation

The previous sections discussed detection of temporal antecedent failure. If a formula is valid on a model M, then using Proposition 5 we can check whether it suffers from TAF. If the formula suffers from TAF we seek for the reason for TAF , if it does not suffer from TAF we would like to convince the user of this as well. In [5] Beer et al. suggested the term *witness* for a trace convincing of the fact the formula does not hold vacuously.

In our case, intuitively, a witness should show that no position is a reason for TAF. Indeed, by Proposition 1, this is an indication that the formula does not suffer from TAF. Recall however that the existence of a secondary reason is not an indication for TAF. However, users may want to get a witness to the fact that a position is not a secondary reason as well.

We thus define two kinds of witnesses. A nonTAF-witness is a trace convincing of the fact that the model does not suffer from TAF with respect to the given RE. A nonReason-witness is a trace convincing of the fact that a given position is not a reason for TAF of r in M. A *full witness* is a set of traces contradicting the fact that there exists a position which is a reason for TAF of r in M. Formally,

Definition 8 (Witnesses). *Let M be a model, r an* RE *and x a position in r.*

- *A run β of a* DTS *D_M is a* nonTAF-witness for r in M iff there exists a prefix α of β such that $L(\alpha) \in [\![r]\!]$
- A run β of a DTS $\mathcal{D}_M$ is a *non*Reason*-witness for r in M with respect to x iff it is a non*TAF*-witness and there exists a prefix α of β such that $L(\alpha) \in [\![r_{\underline{x}|}]\!]$.*

A set $S \subseteq L(\mathcal{D}_M)$ is a full witness for r in $\mathcal{D}_M$ iff for each $x \in pos(r)$, there exists $\beta \in S$ such that β is a nonReason-witness for r in $\mathcal{D}_M$ with respect to x.

The generation of witnesses uses the model-checking ability to generate counter examples. In order to provide a nonTAF-witness we ask the model checker to find a counter example to the fact that the accepting states of the NFA for the given RE are never visited. In order to provide a nonReason-witness for a given position x we ask the model checker to find a counter example to the fact that in addition, whenever the state associated with x is visited, the corresponding letter does not hold. The following proposition states this formally.

Proposition 8. *Let M be a model, r an* RE *and x a position in r.*

- *A counter example for $\mathsf{G}\,(\neg\mathcal{A}_r)$ in $\mathcal{D}_M \,|||\, \mathcal{D}_r$ is a non*TAF*-witness for r in $\mathcal{D}_M$.*
- *A counter example for $\mathsf{G}\,(at\_(x) \rightarrow \neg\ell(x)) \wedge \mathsf{G}\,(\neg\mathcal{A}_r)$ in $\mathcal{D}_M \,|||\, \mathcal{D}_r$ is a non*Reason*-witness for r in $\mathcal{D}_M$ with respect to x.*

4.3 Procedure for Detecting TAF, Its Reasons and Generating Witnesses

Procedure **FindTAFAndGenerateWitness** described below, receives as input a DTS representation $\mathcal{D}_M$ of a model M and an RE r. If M suffers from TAF of r the procedure returns the reasons for TAF (both primary and secondary). If M does not suffer from TAF of r the procedure returns a nonTAF-witness. The procedure does not generate nonReason-witness. It is easy to augment it to produce a nonReason-witness for a given position x (or for all positions x), by adding additional invariant checks as described in Proposition 8.

Procedure 1. Find TAF and Generate Witness()

```
Input: DTS $\mathcal{D}_M$; RE $r$;
Output: set PrimaryList; set SecondaryList; trace WitnessTrace;
PrimaryList := $\emptyset$;
SecondaryList := $\emptyset$;
FindTAFReasons($\mathcal{D}_M$,$r$,$x_\infty$,true,PrimaryList,SecondaryList,$\emptyset$);
if $PrimaryList = \emptyset$ then
    WitnessTrace := ProduceCounterExample($\mathsf{AG}\neg at_{x_\infty}$);
end
```

The procedure **FindTAFReasons** is a recursive procedure that receives as input a DTS representation $\mathcal{D}_M$ of a model M, an RE r, a position x, a flag $primary$ indicating whether a primary or secondary reason is searched for, and three sets $PrimaryList$, $SecondaryList$ and $VisitedPos$ to save sets of positions as implied by the name of the set. At the initial call to the procedure the $primary$ flag is turned on. The position sent at the initial call is x_∞ which is a position added in the construction of the natural NFA (see section A.2 of the full version). This position is added as the last position in order to create a unique last position and simplify the procedure. That is, if r is a natural RE the procedure works on $r \cdot x_\infty$.

The procedure first checks that the given position x was not visited before. If it was visited it returns (line 3) otherwise it updates the set of visited positions accordingly (line 5). It then checks whether the position is reachable (line 6). If it is unreachable then (according to Proposition 5) it is not a reason. It thus calls recursively to each of x's immediate predecessors (lines 20-21). If it is reachable it checks whether it is "exercised" — i.e. whether the corresponding letter holds when expected (line 8). If it is not exercised then (by Proposition 5) a reason is found. The position is inserted to either $PrimaryList$ or $SecondaryList$ according to the value of the $primary$ flag (lines 10-13). The procedure proceeds with recursive calls to the immediate predecessor with the flag $primary$ turned off (line 16-17). Proposition 9 below states the correctness of the procedure.

Proposition 9. *Let $\mathcal{D}_M$ be a* DTS *representation of a model M and r an* RE*. Let PrimaryList, SecondaryList and WitnessTrace be the output of the procedure **FindTAFAndGenerateWitness** on the input r and $\mathcal{D}_M$. Then*

1. *The formula does not suffer from* TAF *in M iff $PrimaryList = \emptyset$.*
2. *PrimaryList holds the set of positions which are a primary reason of* TAF.
3. *SecondaryList holds the set of positions which are a secondary reason of* TAF.
4. *If $PrimaryList = \emptyset$ then WitnessTrace holds a non*TAF*-witness for r in M.*

Procedure 2. FindTAFReasons(FDS $\mathcal{D}_M$,RE r, position x, Bool primary, set PrimaryList, set SecondaryList, set VisitedPos)

```
Bool unreachable, unexercised;
if $x \in VisitedPos$ then
    return;
end
VisitedPos := VisitedPos $\cup\{x\}$
unreachable := $\mathcal{D}_M \models \mathsf{AG}\neg(at\_x)$
if unreachable == false then
    unexercised := $\mathcal{D}_M \models \mathsf{AG}(at\_x \rightarrow \neg\ell(x))$
    if unexercised then
        if primary then
            PrimaryList := PrimaryList $\cup\{x\}$
        else
            SecondaryList := SecondaryList$\cup\{x\}$
        end
    end
    foreach $y \in \mathcal{P}(r,x)$ do
        FindTAFReasons($\mathcal{D}_M$,$r$,$y$,false,PrimaryList,SecondaryList,VisitedPos);
    end
else
    foreach $y \in \mathcal{P}(r,x)$ do
        FindTAFReasons($\mathcal{D}_M$,$r$,$y$,primary,PrimaryList,SecondaryList,VisitedPos);
    end
end
```

5 Discussion

Several definitions of vacuity exist in the literature. The commonly used one is that of Beer et al. [3] cited in the introduction, saying that φ is vacuous in a model M if there exists a sub-formula ψ of φ that *does not affect* the truth value of φ in M. We note that TAF refines vacuity in the sense that if φ suffers from TAF in M it is also vacuous in M according to Beer et al. To see this, let $\varphi = (r \mapsto \Psi)$, and assume that $M \models r \mapsto \Psi$ and r suffers from TAF in M. Thus, the full condition r never occurs in M, and therefore for any formula Ψ', $M \models r \mapsto \Psi'$. Thus Ψ does not affect the truth value of φ in M.

Our method detects TAF by asserting a single invariance (safety) formula, derived from the automaton already built for model checking of the original formula. Thus, it has two advantages over existing methods. First, only *one* formula needs to be checked, as opposed to several formulas — one for each proposition — in other methods [24,1]. Second, it avoids building another automaton for the vacuity formula. The actual effort needed to detect TAF depends on the model checking method used in practice. In the worst case it amounts to another model checking run of an invariance formula. However, when the reachable state space is computed, as done by BDD-based model checkers such as SMV [25], TAF is detected by intersecting the BDD of the invariance formula with that of the reachable states. This intersection is easily performed, and never takes more than a few seconds, even for very large models. TAF is therefore detected with almost no overhead on the model checking process.

TAF detection is implemented in the IBM model checking tool RuleBase [21] and serves as the major vacuity detection algorithm of the tool (applied to all formulas that can be translated into suffix implication form, which forms the vast majority of formulas written in practice [8]). Unlike evidence regarding other types of vacuity, experience shows that temporal antecedent failure always indicates a real problem in the model, environment or formula under verification. Evidence in the other direction are less clear. Although non-TAF vacuities are many times considered non-problem by the users, this is not always the case. We thus propose a two button approach. Button 1 (the default), would check for TAF that, on the one hand is guaranteed to be important and on the other hand can be detected efficiently. If the user is interested in a more comprehensive check, with the risk of it producing non-real problems and taking a longer time, she can then choose button 2, that would perform the desired comprehensive check.

References

1. Armoni, R., Fix, L., Flaisher, A., Grumberg, O., Piterman, N., Tiemeyer, A., Vardi, M.: Enhanced vacuity detection in linear temporal logic. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 368–380. Springer, Heidelberg (2003)
2. Beatty, D., Bryant, R.: Formally verifying a microprocessor using a simulation methodology. In: Design Automation Conference, pp. 596–602 (1994)
3. Beer, I., Ben-David, S., Eisner, C., Rodeh, Y.: Efficient detection of vacuity in ACTL formulas. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 279–290. Springer, Heidelberg (1997)
4. Beer, I., Ben-David, S., Eisner, C., Rodeh, Y.: Efficient detection of vacuity in temporal model checking. Formal Methods in System Design 18(2), 141–163 (2001)
5. Beer, I., Ben-David, S., Landver, A.: On-the-fly model checking of RCTL formulas. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 184–194. Springer, Heidelberg (1998)
6. Ben-David, S., Fisman, D., Ruah, S.: Automata construction for regular expressions in model checking (IBM research report H-0229) (2004)
7. Ben-David, S., Fisman, D., Ruah, S.: Embedding finite automata within regular expressions. In: Margaria, T., Steffen, B. (eds.) 1st International Symposium on Leveraging Applications of Formal Methods, Springer (2004)
8. Ben-David, S., Fisman, D., Ruah, S.: The safety simple subset. In: Ur, S., Bin, E., Wolfsthal, Y. (eds.) Hardware and Software, Verification and Testing. LNCS, vol. 3875, pp. 14–29. Springer, Heidelberg (2006)
9. Ben-David, S., Fisman, D., Ruah, S.: Temporal antecedent failure: Refining vacuity (IBM research report H-0252) (2007)
10. Berry, G., Sethi, R.: From regular expressions to deterministic automata. Theoretical Computer Science 48(1), 117–126 (1986)
11. Boul'e, M., Zilic, Z.: Efficient automata-based assertion-checker synthesis of psl properties. In: HLDVT06. Workshop on High LevelDesign, Validation, and Test (2006)
12. Bustan, D., Fisman, D., Havlicek, J.: Automata construction for PSL. Technical Report MCS05-04, The Weizmann Institute of Science (2005)
13. Bustan, D., Flaisher, A., Grumberg, O., Kupferman, O., Vardi, M.Y.: Regular vacuity. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, Springer, Heidelberg (2005)

14. Chechik, M., Gurfinkel, A., Gheorghiu, M.: Finding environmental guarantees. In: FASE 2007 (2007)
15. Clarke, E., Emerson, E.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logics of Programs. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
16. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. The MIT Press, Cambridge (2000)
17. Dong, Y., saran Starosta, B., Ramakrishnan, C., Smolka, S.A.: Vacuity checking in the modal mu-claculus. In: Kirchner, H., Ringeissen, C. (eds.) AMAST 2002. LNCS, vol. 2422, pp. 147–162. Springer, Heidelberg (2002)
18. Eisner, C., Fisman, D.: A Practical Introduction to PSL. Springer (2006)
19. Gurfinkel, A., Chechik, M.: Extending extended vacuity. In: Hu, A.J., Martin, A.K. (eds.) FMCAD 2004. LNCS, vol. 3312, pp. 306–321. Springer, Heidelberg (2004)
20. Gurfinkel, A., Chechik, M.: How vacuous is vacuous? In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 451–466. Springer, Heidelberg (2004)
21. IBM's Model Checker RuleBase. http://www.haifa.il.ibm.com/projects/verification/rb_homepage/index.html
22. Annex E of IEEE Standard for SystemVerilog — Unified Hardware Design, Specification, and Verification Language. IEEE Std 1800^{TM}-2005
23. IEEE Standard for Property Specification Language (PSL). IEEE Std 1850^{TM}-2005
24. Kupferman, O., Vardi, M.: Vacuity detection in temporal model checking. STTT 4(2), 224–233 (2003)
25. McMillan, K.: Symbolic Model Checking. Kluwer Academic Publishers, Dordrecht (1993)
26. Namjoshi, K.S.: An efficiently checkable, proof-based formulation of vacuity in model checking. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 57–69. Springer, Heidelberg (2004)
27. Purandare, M., Somenzi, F.: Vacuum cleaning CTL formulae, pp. 485–499 (July 2002)
28. Quielle, J., Sifakis, J.: Specification and verification of concurrent systems in cesar. In: 5th International Symposium on Programming (1982)
29. Samer, M., Veith, H.: Parametrized vacuity. In: Hu, A.J., Martin, A.K. (eds.) FMCAD 2004. LNCS, vol. 3312, pp. 322–336. Springer, Heidelberg (2004)
30. Tzoref, R., Grumberg, O.: Automatic refinement and vacuity detection for symbolic trajectory evaluation. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 190–204. Springer, Heidelberg (2006)

Author Index