Paul Gastin
François Laroussinie (Eds.)

LNCS 6269

CONCUR 2010 – Concurrency Theory

21st International Conference, CONCUR 2010
Paris, France, August/September 2010
Proceedings

Springer

Lecture Notes in Computer Science 6269

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Advanced Research in Computing and Software Science

Subline of Lectures Notes in Computer Science

Paul Gastin   François Laroussinie (Eds.)

CONCUR 2010 – Concurrency Theory

21st International Conference, CONCUR 2010
Paris, France, August 31 - September 3, 2010
Proceedings

Springer

Volume Editors

Paul Gastin
Laboratoire Spécification et Vérification (LSV)
ENS de Cachan & CNRS
94235 Cachan cedex, France
E-mail: paul.gastin@lsv.ens-cachan.fr

François Laroussinie
LIAFA, Université Paris Diderot
75205 Paris cedex 13, France
E-mail: francois.laroussinie@liafa.jussieu.fr

Library of Congress Control Number: 2010933357

CR Subject Classification (1998): D.2, F.3, C.2, D.3, H.4, D.2.4

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

ISSN 0302-9743
ISBN-10 3-642-15374-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-15374-7 Springer Berlin Heidelberg New York

springer.com

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

Preface

This volume contains the proceedings of the 21st Conference on Concurrency Theory (CONCUR 2010), held in Paris, at Université Denis Diderot, August 31–September 3, 2010. CONCUR 2010 was organized by CNRS, INRIA, ENS Cachan and the Université Denis Diderot.

The purpose of the CONCUR conference is to bring together researchers, developers, and students in order to advance the theory of concurrency and promote its applications. The principal topics include *basic models of concurrency* such as abstract machines, domain theoretic models, game theoretic models, process algebras, and Petri nets; *logics for concurrency* such as modal logics, probabilistic and stochastic logics, temporal logics, and resource logics; *models of specialized systems* such as biology-inspired systems, circuits, hybrid systems, mobile and collaborative systems, multi-core processors, probabilistic systems, real-time systems, service-oriented computing, and synchronous systems; *verification and analysis techniques for concurrent systems* such as abstract interpretation, atomicity checking, model checking, race detection, pre-order and equivalence checking, run-time verification, state–space exploration, static analysis, synthesis, testing, theorem proving, and type systems; *related programming models* such as distributed, component-based, object-oriented, and Web services.

This edition of the conference attracted 107 submissions. We wish to thank all their authors for their interest in CONCUR 2010. After careful discussions, the Program Committee selected 35 papers for presentation at the conference. Each submission was refereed by three reviewers, who delivered detailed and insightful comments and suggestions. The conference Chairs warmly thank all the members of the Program Committee and all their referees for the excellent support they gave, as well as for the friendly and constructive discussions. We would also like to thank the authors for having revised the papers to address the comments and suggestions by the referees.

The conference program was enriched by the outstanding invited talks by:

- Frank S. de Boer (CWI, Amsterdam, The Netherlands)
- Maurice Herlihy (Brown University, Providence, USA)
- Holger Hermanns (Saarland University, Saarbrücken, Germany)

- Anca Muscholl (LaBRI, University Bordeaux 1, France)
- Vladimiro Sassone (ECS, University of Southampton, UK)

The conference this year was co-located with the 17th International Symposium on Temporal Representation and Reasoning (TIME 2010). Additionally, CONCUR 21010 included the following satellite workshops:

- Structural Operational Semantics (SOS 2010)
- Expressiveness in Concurrency (EXPRESS 2010)
- Security in Concurrency (SecCo 2010)
- Foundations of Interface Technologies (FIT 2010)
- GASICS Workshop on Games for Design, Verification and Synthesis
- Foundations of Coordination Languages and Software Architectures (FOCLASA 2010)
- International Workshop on Distributed Open and Timed Systems (DOTS 2010)
- Young Researchers Workshop on Concurrency Theory (YR-CONCUR)

We would like to thank all the people who contributed to the success of CONCUR 2010, in particular the Organizing Committee and the administrative staff. We gratefully acknowledge the generous support from Digiteo, Région Île de France, CNRS, INRIA, ENS Cachan, the Ministère de l'enseignement supérieur et de la recherche, and the Université Denis Diderot - Paris 7. We are also grateful to Andrei Voronkov for his excellent EasyChair conference system, which we used for the eletronic submission of papers, the refereeing process, and the Program Committee work.

June 2010

Paul Gastin
François Laroussinie

Conference Organization

Program Chairs

Paul Gastin	LSV, ENS Cachan, France
François Laroussinie	LIAFA, Université Denis Diderot - Paris 7, France

Program Committee

Martín Abadi	UC Santa Cruz and Microsoft Research, CA, USA
Parosh Abdulla	Uppsala University, Sweden
Jos Baeten	Eindhoven University of Technology, The Netherlands
Steffen van Bakel	Imperial College London, UK
Julian Bradfield	University of Edinburgh, UK
Luís Caires	Universidade Nova de Lisboa, Portugal
Luca Cardelli	Microsoft Research Cambridge, UK
Vincent Danos	University of Edinburgh, UK
Daniele Gorla	Università di Roma “La Sapienza”, Italy
Anna Ingólfsdóttir	Reykjavik University, Iceland
Petr Jančar	Technical University of Ostrava, Czech Republic
Joost-Pieter Katoen	RWTH Aachen University, Germany and University of Twente, The Netherlands
Kim Larsen	Aalborg University, Denmark
Nancy Lynch	Massachusetts Institute of Technology, USA
Ugo Montanari	Università di Pisa, Italy
Markus Müller-Olm	Münster University, Germany
K. Narayan Kumar	Chennai Mathematical Institute, India
Catuscia Palamidessi	LIX, INRIA, France
Prakash Panangaden	McGill School of Computer Science, Canada
Wojciech Penczek	IPI PAN, University of Podlasie, Poland
Shaz Qadeer	Microsoft Research Redmond, WA, USA
Jan Rutten	CWI, The Netherlands
Roberto Segala	Università di Verona, Italy
P.S. Thiagarajan	National University of Singapore
Walter Vogler	University of Augsburg, Germany

Steering Committee

Roberto Amadio	PPS, Université Paris Diderot - Paris 7, France
Jos Baeten	Eindhoven University of Technology, The Netherlands
Eike Best	Carl von Ossietzky Universität Oldenburg, Germany
Kim Larsen	Aalborg University, Denmark
Ugo Montanari	Università di Pisa, Italy
Scott Smolka	SUNY, Stony Brook University, USA

Organizing Committee

Benedikt Bollig	LSV, CNRS, France
Stefan Haar	LSV, INRIA, France
Florian Horn	LIAFA, CNRS, France
Stefan Schwoon	LSV, INRIA, France
Olivier Serre	LIAFA, CNRS, France
Mihaela Sighireanu	LIAFA, Université Denis Diderot - Paris 7, France

External Reviewers

Roberto M. Amadio
Suzana Andova
Tadashi Araragi
Farhad Arbab
Mohamed Faouzi Atig
Franco Barbanera
Massimo Bartoletti
Emmanuel Beffara
Nick Benton
Josh Berdine
Marco Bernardo
Nathalie Bertrand
Dietmar Berwanger
Chiara Bodei
Frank S. de Boer
Bernard Boigelot
Benedikt Bollig
Filippo Bonchi
Tomáš Brázdil
Franck van Breugel
Václav Brožek
Roberto Bruni
Marco Carbone
Pietro Cenciarelli
Taolue Chen
Yu-Fang Chen
Corina Cîrstea
Dave Clarke
Thomas Colcombet
Alejandro Cornejo
Flavio Corradini
Pedro R. D'Argenio
Ugo de'Liguoro
Stéphane Demri

Yuxin Deng
Rocco De Nicola
Mariangiola Dezani
Maria Rita Di Berardini
Alessandro D'Innocenzo
Lucas Dixon
Laurent Doyen
Deepak D'Souza
Jérémy Dubreil
Ross Duncan
Tayfun Elmas
Javier Esparza
Yuan Feng
Jérôme Feret
Bernd Finkbeiner
Cormac Flanagan
Vojtěch Forejt
Martin Fränzle
Carsten Fritz
Fabio Gadducci
Philippa Gardner
Simon Gay
Blaise Genest
Nargess Ghahremani
Marco Giunti
Stefan Göller
Alexey Gotsman
Carlos Gregorio-Rodríguez
Marcus Groesser
Julian Gutierrez
Peter Habermehl
Serge Haddad
Frédéric Haziza
Thomas Hildebrandt
Peter Höfner
Espen Højsgaard
Lukáš Holík
Chih-Duo Hong
Hans Hüttel
Radha Jagadeesan
Agata Janowska
Mariusz Jarocki
Christophe Joubert
Bartek Klin
Michał Knapik
Alexander Knapp
Sophia Knight
Natallia Kokash
Barbara König
Beata Konikowska
Martin Kot
Christian Krause
Jean Krivine
Dietrich Kuske
Akash Lal
Peter Lammich
Mikkel Larsen Pedersen
Jérôme Leroux
Jean-Jacques Lévy
Xinxin Liu
Kamal Lodaya
Michele Loreti
Étienne Lozes
Bas Luttik
Pasquale Malacaria
Radu Mardare
Nicolas Markey
Jasen Markovski
Tomáš Masopust
Mieke Massink
Richard Mayr
Antoni Mazurkiewicz
Damiano Mazza
Frank McSherry
Alexandru Mereacre
Massimo Merro
Roland Meyer
Marino Miculan
Rémi Morin
Madan Musuvathi

Wojtek Nabiałek
Sebastian Nanz
Daniel Neider
Calvin Newport
Dejan Ničković
Artur Niewiadomski
Thomas Noll
Gethin Norman
Jan Obdržálek
Petur Olsen
Rotem Oshman
Luca Padovani
David Parker
Jorge A. Perez
Iain Phillips
Sophie Pinchinat
Nir Piterman
Sanjiva Prasad
Rosario Pugliese
Paola Quaglia
Sriram Rajamani
R. Ramanujam
António Ravara
Ahmed Rezine
Pau Ruet
Joshua Sack
Arnaud Sangnier
Ivano Salvo
Davide Sangiorgi
Zdeněk Sawa
Jens-Wolfhard Schicke
Alan Schmitt
Sven Schneider
Stefan Schwoon
Anil Seth
Jaroslav Ševčík
Ali Sezgin
Mihaela Sighireanu
Alexandra Silva
Robert Simmons
Anu Singh
Satnam Singh
Paweł Sobociński
Ana Sokolova
Monika Solanki
Jeremy Sproston
Jiří Srba
Christian Stahl
Jan Strejček
Georg Struth
Rong Su
Kohei Suenaga
Jun Sun
S.P. Suresh
Maciej Szreter
Andrzej Tarlecki
Claus Thrane
Francesco Tiezzi
Frank Valencia
Björn Victor
Hugo Vieira
Saleem Vighio
Maria Grazia Vigliotti
Erik de Vink
Marc Voorhoeve
Igor Walukiewicz
Alexander Wenner
Józef Winkowski
Verena Wolf
Weng-Fai Wong
Bożena Woźna-Szcześniak
Tobias Wrigstad
Alex Yakovlev
Nobuko Yoshida
Kenneth Yrke Jørgensen
Gianluigi Zavattaro
Marc Zeitoun
Lijun Zhang

Table of Contents

Dating Concurrent Objects: Real-Time Modeling and Schedulability Analysis*

Frank S. de Boer[1,2], Mohammad Mahdi Jaghoori[1,2], and Einar Broch Johnsen[3]

[1] CWI, Amsterdam, The Netherlands
[2] LIACS, Leiden, The Netherlands
[3] University of Oslo, Norway

Abstract. In this paper we introduce a real-time extension of the concurrent object modeling language Creol which is based on duration statements indicating best and worst case execution times and deadlines. We show how to analyze schedulability of an abstraction of real-time concurrent objects in terms of timed automata. Further, we introduce techniques for testing the conformance between these behavioral abstractions and the executable semantics of Real-Time Creol in Real-Time Maude.

As a case study we model and analyze the schedulability of thread pools in an industrial communication platform.

1 Introduction

In the object-oriented modeling language Creol [9,3], objects are concurrent; i.e., conceptually, each object encapsulates its own processor. Therefore, each object has a single thread of execution. Creol objects communicate by asynchronous message passing. The message queue is implicit in the objects. Furthermore, the scheduling policy is underspecified; i.e., messages in the queue are processed in a nondeterministic order. The running method can voluntarily release the processor using special primitives, allowing another message to be scheduled. For example, a method can test whether an asynchronous call has been completed, and if not, release the processor; thus modeling synchronous calls.

In this paper we extend Creol with real-time information about the deadlines of messages and the best and worst execution times of the (sequential) control statements. We formalize the semantics of Real-Time Creol with respect to given intra-object scheduling policies in the real-time extension of Maude [5]. This formalization of a Real-Time Creol model provides a refinement of the underlying untimed model in the sense that it only restricts its behaviors.

Schedulability analysis. In general analyzing schedulability of a real time system consists of checking whether all tasks are accomplished within their deadlines.

* This research is partly funded by the EU projects IST-33826 CREDO: Modeling and Analysis of Evolutionary Structures for Distributed Services (`http://credo.cwi.nl`) and FP7-231620 HATS: Highly Adaptable and Trustworthy Software using Formal Methods (`http://www.hats-project.eu`).

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 1–18, 2010.

We employed automata theory in our previous work [7,8] to provide a high-level framework for modular schedulability analysis of concurrent objects. In order to analyze the schedulability of an open system of concurrent objects, we need some assumptions about the real-time arrival patterns of the incoming messages; in our framework, this is contained in the timed automata [1] modeling the *behavioral interface* of the open system. A behavioral interface captures the overall real-time input/output behavior of an object while abstracting from its detailed implementation in its methods; a deadline is assigned to each message specifying the time before which the corresponding method has to be completed. Further, we use timed automata to describe an abstraction of the system of objects itself including its message queues and a given scheduling policy (e.g., Earliest Deadline First). The analysis of the schedulability of an open system of concurrent objects can then be reduced to model-checking a timed automaton describing the interactions between the behavioral abstraction of the system and its behavioral interface (representing the environment).

Conformance. We test conformance between the Real-Time Creol model of an open system of concurrent objects and its behavioral abstraction in timed automata with respect to a given behavioral interface. Our method is based on generating a timed trace (i.e., a sequence of time-stamped messages) from the automaton constructed from its behavioral abstraction and interface. Using model-checking techniques we next generate for each time specified in the trace additional real-time information about all possible observable messages. This additional information allows us to find *counter-examples* to the conformance. To do so, we use the Real-Time Maude semantics as a language interpreter to execute the real-rime Creol model driven by the given trace. Then we look for counter-examples by incrementally searching the execution space for possible timed observations that are not covered in the extended timed trace.

Case Study. Thread pools are an important design pattern used frequently in industrial practice to increase the throughput and responsiveness of software systems, as for instance in the ASK system [2]. The ASK system is an industrial communication platform providing mechanisms for matching users requiring information or services with potential suppliers. A thread pool administrates a collection of computation units referred to as threads and assigns tasks to them. This administration includes dynamic creation or removal of such units, as well as scheduling the tasks based on a given strategy like ‘first come first served’ or priority based scheduling.

The abstraction from the internal message queue of each object and the related scheduling policies is one of the most important characteristics of Creol which allows for abstractly modeling a variety of thread pools. In this paper, we give an example of an abstract model in Creol of a basic pool where the threads share the task queue. The shared task queue is naturally represented *implicitly* inside a Creol object (called a resource-pool) that basically forwards the queued tasks to its associated threads also represented as Creol objects. We associate real-time information to the tasks concerning their deadlines and best and worst case execution times.

We perform schedulability analysis on a network of timed automata constructed from the automata abstraction and behavioral interface of the thread pool model in order to verify whether tasks are performed within their deadlines. In the context of the ASK system, schedulability ensures that the response times for service requests are always bounded by the deadlines. We use Uppaal [12] for this purpose. Further, we test conformance between the Real-Time Creol model of a thread pool and its behavioral abstractions as described above.

Related work. We extend Creol with explicit scheduling strategies and a duration statement to specify execution delays. Creol$_{\mathrm{RT}}$ [11] is another real-time extension of Creol with clocks and time-outs. Our work follows a descriptive approach to specifying real-time information suitable for schedulability analysis, whereas the prescriptive nature of time in Creol$_{\mathrm{RT}}$ can affect the functional behavior of an object.

Schedulability analysis in this paper can be seen as the continuation of our previous work [7] on modular analysis of a single-threaded concurrent object with respect to its behavioral interface. In this paper, we extend the schedulability analysis to an open system of concurrent objects in a way similar to [4].

The work of [6,10] is based on extracting automata from code for schedulability analysis. However, they deal with programming languages and timings are usually obtained by profiling the real system. Our work is applied on high-level models. Therefore, our main focus is on studying different scheduling policies and design decisions.

We test conformance between a Creol implementation and abstract automata models. Our notion of conformance is similar to **tioco** introduced by Schmaltz and Tretmans [16,15], but we do not directly work with timed input/output transition systems; an innovation of our work is dealing with conformance between different formalisms, namely Creol semantics in rewrite logic on one hand and timed automata on the other hand. Furthermore, we focus on generating counterexamples during testing along the lines of our previous work [8], which is novel in testing.

Outline. The real-time extension of the concurrent object language Creol is explained in Section 2. As explained in Section 3, abstract models of concurrent objects, specified in timed automata, are analyzed to be schedulable. To be able to argue about the schedulability of Real-Time Creol models, we need to establish conformance between our Creol and automata models; this is explained in Section 4. We conclude in section 5.

2 Concurrent Objects in Real-Time Creol

Creol is an abstract behavioral modeling language for distributed active objects, based on asynchronous method calls and processor release points. In Creol, objects conceptually have dedicated processors and live in a distributed environment with asynchronous and unordered communication between objects. Communication is between named objects by means of asynchronous method calls;

Syntactic categories.	*Definitions.*
g in Guard	$IF ::= \textbf{interface}\ I\ \{\ [\overline{Sg}]\ \}$
s in Stmt	$CL ::= \textbf{class}\ C\ [(\overline{I\ x})]\ [\textbf{implements}\ \overline{I}]\ \{\ [\overline{I\ x};]\ \overline{M}\}$
x in Var	$Sg ::= I\ m\ ([\overline{I\ x}])$
e in Expr	$M ::= Sg\ \{\ [\overline{I\ x};]\ s\ \}$
o in ObjExpr	$g ::= b \mid x? \mid g \wedge g \mid g \vee g$
b in BoolExpr	$s ::= x := e \mid x := e.\textbf{get} \mid \textbf{skip} \mid \textbf{release} \mid \textbf{await}\ g \mid \textbf{return}\ e$
d in Time	$\mid s; s \mid [o]!m(\overline{e}, d) \mid \textbf{if}\ b\ \textbf{then}\ \{\ s\ \}\ \textbf{else}\ \{\ s\ \} \mid \textbf{duration}(d, d)$
	$e ::= x \mid o \mid b \mid \textbf{new}\ \mathsf{C}\ (\overline{e}) \mid [o]!m(\overline{e}, d) \mid \textbf{this} \mid \textbf{deadline}$

Fig. 1. Syntax of the Real-Time Creol kernel. Terms such as $\overline{e}$ and $\overline{x}$ denote lists over the corresponding syntactic categories and square brackets denote optional elements.

these may be seen as triggers of concurrent activity, resulting in new activities (tasks) in the called object. Objects are dynamically created instances of classes, declared attributes are initialized to some arbitrary type-correct values. An optional *init* method may be used to redefine the attributes during object creation. An object has a set of tasks to be executed, which stem from method activations. Among these, at most one task is *active* and the others are *suspended* on a task queue. The scheduling of tasks is by default non-deterministic, but controlled by *processor release points* in a cooperative way. Creol is strongly typed: for well-typed programs, invoked methods are supported by the called object (when not `null`), such that formal and actual parameters match. In this paper, programs are assumed to be well-typed. This section introduces *Real-Time Creol*, explaining Creol constructs (for further details, see, e.g., [9,3]) and their relation to real-time scheduling policies.

Figure 1 gives the syntax for a kernel of Real-Time Creol, extending a subset of Creol (omitting, e.g., inheritance). A *program* consists of interface and class definitions and a `main` method to configure the initial state. Let C, I, and m be in the syntactic category of Names. IF defines an interface with name I and method signatures Sg. A class implements a list $\overline{I}$ of interfaces, specifying types for its instances. CL defines a class with name C, interfaces $\overline{I}$, class parameters and state variables $\overline{x}$ (of types $\overline{I}$), and methods $\overline{M}$. (The *attributes* of the class are both its parameters and state variables.) A method signature Sg declares the return type I of a method with name m and formal parameters $\overline{x}$ of types $\overline{I}$. M defines a method with signature Sg and a list of local variable declarations $\overline{x}$ of types $\overline{I}$ and a statement s. Statements may access class attributes, locally defined variables, and the method's formal parameters. Statements for assignment $x := e$, sequential composition $s_1; s_2$, **skip**, **if**, **while**, and **return** e are standard. The statement **release** unconditionally releases the processor by suspending the active task. In contrast, the guard g controls processor release in the statement **await** g, and consists of Boolean conditions which may contain return tests $x?$ (see below). If g evaluates to false, the current task is *suspended* and the execution thread becomes idle. When the execution thread is idle, an enabled task may be selected from the pool of suspended tasks by means of a user-provided scheduling policy.

Expressions e include declared variables x, object identifiers o, Boolean expressions b, and object creation **new** $C(\overline{e})$. As usual, the reserved read-only variable **this** refers to the identifier of the object. Note that remote access to attributes is not allowed. (The full language includes a functional expression language with standard operators for data types such as strings integers lists, sets, maps, and tuples. These are omitted in the core syntax, and explained when used in the examples.)

Time. In Real-Time Creol, the local passage of time is expressed in terms of **duration** statements. We consider a dense time model represented by the sort Time which ranges over non-negative real numbers and is totally ordered by the less-than operator. Furthermore, we denote by ∞ a term of sort Time such that for all $t_1, t_2 \neq \infty$, $t_1 + t_2 < \infty$. The statement **duration**(b, w) expresses the passage of time, given in terms of an interval between the best case b and the worst case w (assuming $b \leq w$). All other statements are assumed to be instantaneous, except the **get** statement which lets time pass while it is blocking (see below).

Communication in Real-Time Creol is based on asynchronous method calls, denoted by expressions $o!m(\overline{e}, d)$, and future variables. (Local calls are written $!m(\overline{e}, d)$.) Thus, after making an asynchronous method call $x := o!m(\overline{e}, d)$, the caller may proceed with its execution without blocking on the method reply. Here x is a future variable, o is an object expression, $\overline{e}$ are (data value or object) expressions, and d is a *deadline* for the method invocation. This deadline specifies the relative time before which the corresponding method should be scheduled and executed. The local variable **deadline** refers to the *remaining permitted execution time* of the current method activation. We assume that message transmission is instantaneous, so the deadline expresses the time until a reply is received; i.e., it corresponds to an *end-to-end* deadline. As usual, if the return value of a call is of no interest, the call may occur as a statement. The future variable x refers to a return value which has yet to be computed. There are two operations on future variables, which control synchronization in Creol. First, the guard **await** $x?$ suspends the active task unless a return to the call associated with x has arrived, allowing other tasks in the object to execute. Second, the return value is retrieved by the expression x.**get**, which blocks all execution in the object until the return value is available. Standard usages of asynchronous method calls include the statement sequence $x := o!m(\overline{e}, d);\ v := x$.**get** which encodes a *blocking call*, abbreviated $v := o.m(\overline{e}, d)$ (often referred to as a synchronous call), and the statement sequence $x := o!m(\overline{e}, d)$; **await** $x?$; $v := x$.**get** which encodes a non-blocking, *preemptible call*, abbreviated **await** $v := o.m(\overline{e}, d)$.

2.1 Object-Oriented Modeling of Thread-Pools

Figure 2 shows a Creol model of a thread pool. The model defines a `Thread` class and the `ResourcePool` class. The task list is modeled implicitly in terms of the message queue of an instance of the `ResourcePool` class. The variable

```
class Thread(ResourcePool myPool) implements Thread {
  Void run() { myPool!finish(this) }
  Void start() { skip; duration(5,6); myPool!finish(this) }
}

class ResourcePool(Int size) implements ResourcePool {
  Set[Thread] pool;

  Void init() { Thread thr; pool := {};
    while (size>0) { thr := new Thread(this); size := size-1 }
  }
  Void invoke() {
    Thread thread; await ¬isempty(pool);
    thread := choose(pool); pool := remove(pool,thread);
    thread!start(deadline)
  }
  Void finish (Thread thr) { pool := add(pool,thr) }
}
```

Fig. 2. The thread pool

`size` represents the number of available threads, i.e., instances of the `Thread` class. The variable `pool` is used to hold a reference to those threads that are currently not executing a task. Tasks are modeled in terms of the method start inside the `Thread` class. For our analysis the functional differences between tasks is irrelevant, so the method is specified in terms of its duration only and a subsequent call to the method `finish` of the `ResourcePool` object which adds that thread to its pool of available threads.

Tasks are generated (by the environment) with (asynchronous) calls of the `invoke` method of the `ResourcePool` object. In case there are no available threads, the execution of the `invoke` method suspends by the execution of the **`await`** statement which releases control (so that a call of the `finish` method can be executed). When multiple tasks are pending and a thread becomes available, the scheduling strategy of the `ResourcePool` object determines which task should be executed next when the current task has been completed.

2.2 Real-Time Execution in Real-Time Maude

Real-Time Maude [14] defines real-time rewrite theories (Σ, E, IR, TR), where:

- (Σ, E) is a theory in membership equational logic [13] with a signature Σ and a set E of conditional equations. The system's state space is defined as an algebraic data type in (Σ, E), which is assumed to contain a specification of a sort `Time` capturing the (dense or discrete) time domain.
- IR is a set of labeled conditional *instantaneous rewrite rules* specifying the system's local transitions, written **`crl`** $[l] : t \longrightarrow t'$ **`if`** *cond*, where l is a name for the rule. Such a rule specifies a one-step transition (in zero-time) from an instance of a pattern t to the corresponding instance of a pattern t', provided the condition *cond* holds. As usual in rewriting logic [13], the rules are applied modulo the equations in E.

- TR is a set of *timed rewrite rules* (or tick rules) capturing the progression of time, written **crl** $[l] : \{t\} \longrightarrow \{t'\}$ **in time** τ **if** *cond* where τ is a term of sort `Time` which denotes the duration of the rewrite. Observe that `{_}` is the built-in constructor of sort `GlobalSystem`, so tick rules apply to the *entire* state of the system which allows time to advance uniformly.

Initial states must be ground terms of sort `GlobalSystem`, which reduce to terms of the form `{t}` by the equations in E. The form of the tick rules then ensures that time advances uniformly throughout the system. Real-time rewrite theories are executable under reasonable assumptions and Real-Time Maude provides different analysis methods [14]. For example, timed "fair" rewrite simulates one behavior of the system up to a certain duration and is written

```
(tfrew t in time ≤ τ .)
```

for an initial state `t` and a ground term τ of sort `Time`. Furthermore, timed search searches breadth-first for states that are reachable from a given initial state `t` within time τ, match a search pattern, and satisfy a search condition. The command which searches for one state satisfying the search criteria is written

```
(tsearch [1] t ⟶* pattern such that cond in time ≤ τ .)
```

Creol's semantics in Maude. Creol has a semantics defined in Rewriting logic [13] which can be used directly as a language interpreter in Maude [5]. The semantics is explained in detail in [9] and can be used for the analysis of Creol programs. In this section we focus on the extension of Creol's semantics in order to define a semantics for Real-Time Creol in Real-Time Maude.

The state space of Creol's operational semantics is given by terms of the sort `Configuration` which is a set of objects, messages, and futures. The empty configuration is denoted `none` and whitespace denotes the associative and commutative union operator on configurations. Objects are defined as tuples

$$\langle o,\ a,\ q\ \rangle$$

where o is the identifier of the object, a is a map which defines the values of the attributes of the object, and q is the task queue. Tasks are of sort `Task` and consist of a statement s and the task's local variables l. We denote by $\{l|s\} \circ q$ the result of appending the task $\{l|s\}$ to the queue q. For a given object, the first task in the queue is the *active task* and the first statement of the active task to be executed is called the *active statement*.

Let σ and σ' be maps, x a variable name, and v a value. Then $\sigma(x)$ denotes the lookup for the value of x in σ, $\sigma[x \mapsto v]$ the update of σ such that x maps to v, $\sigma \circ \sigma'$ the composition of σ and σ', and $\mathrm{dom}(\sigma)$ the domain of σ. Given a mapping, we denote by $[\![e]\!]^c_\sigma$ the evaluation of an expression e in the state given by σ and the global configuration c (the latter is only used to evaluate the polling of futures; e.g., **`await`** `x?`).

Rewrite rules execute statements in the active task in the context of a configuration, updating the values of attributes or local variables as needed. For an active task $\{l \mid s\}$, these rules are defined inductively over the statement s.

$$\mathbf{rl}\ [\mathit{skip}]:\ \langle o,\ a,\ \{l \mid \mathtt{skip};s\}\circ q\rangle \longrightarrow \langle o,\ a,\ \{l \mid s\}\circ q\rangle\ .$$

$$\begin{array}{l}\mathbf{rl}\ [\mathit{assign}]:\ \langle o,\ a,\ \{l \mid x{:=}e;s\}\circ q\rangle\\ \longrightarrow \mathbf{if}\ x \in \mathrm{dom}(l)\ \mathbf{then}\ \langle o,\ a,\ \{l[x \mapsto [\![e]\!]^{none}_{aol}] \mid s\}\circ q\rangle\\ \qquad\qquad\qquad\qquad \mathbf{else}\ \langle o,\ a[x \mapsto [\![e]\!]^{none}_{aol}],\ \{l \mid s\}\circ q\rangle\ \mathbf{fi}\ .\end{array}$$

$$\mathbf{rl}\ [\mathit{release}]:\langle o,\ a,\ \{l \mid \mathtt{release};s\}\circ q\rangle \longrightarrow \langle o,\ a,\ \mathtt{schedule}(\{l \mid s\},q)\rangle\ .$$

$$\begin{array}{l}\mathbf{crl}\ [\mathit{await1}]:\ \{\langle o,\ a,\ \{l \mid \mathtt{await}\ e;s\}\circ q\rangle\ c\}\\ \longrightarrow \{\langle o,\ a,\ \{l \mid s\}\circ q\rangle\ c\}\ \mathbf{if}\ [\![e]\!]^{c}_{aol}\ .\end{array}$$

$$\begin{array}{l}\mathbf{crl}\ [\mathit{await2}]:\ \{\langle o,\ a,\{l \mid \mathtt{await}\ e;s\}\circ q\rangle\ c\}\\ \longrightarrow \{\langle o,\ a,\ \{l \mid \mathtt{release;await}\ e;s\}\circ q\rangle\ c\}\ \mathbf{if}\ \neg[\![e]\!]^{c}_{aol}\ .\end{array}$$

Fig. 3. The semantics of Creol in Maude

Some (representative) rules are presented in Figure 3. Rule *skip* shows the general set-up, where a **skip** statement is consumed by the rewrite rule. Rule *assign* updates either the local variable or the attribute x with the value of an expression e evaluated in the current state. The suspension of tasks is handled by rule *release*, which places the active task in the task queue. The auxiliary function `schedule` in fact defines the (local) *task scheduling policy* of the object, for example first in first out (FIFO) or earliest deadline first (EDF). Rules *await1* and *await2* handle conditional suspension points.

Real-Time Creol's semantics in Real-Time Maude. The rewrite rules of the Real-Time Creol semantics are given in Figure 4. The first rule ensures that a **duration** statement may terminate only if its best case execution time has been reached. In order to facilitate the conformance testing discussed below, we define a global clock `clock(t)` in the configurations (where `t` is of sort `Time`) to time-stamp observable events. These observables are the invocation and return of method calls. Rule *async-call* emits a message to the callee $[\![e]\!]^{none}_{(a\circ l)}$ with method m, actual parameters $[\![\overline{e}]\!]^{none}_{(a\circ l)}$ including the deadline, a fresh future identifier n, which will be bound to the task's so-called `destiny` variable [3], and, finally, a time stamp t. In the (method) *activation* rule, the function `task` transforms such a message into a task which is placed in the task queue of the callee by means of the scheduling function `schedule`. The function `task` creates a map which assigns the values of the actual parameters to the formal parameters (which includes the `deadline` variable) and which assigns the future identity to the `destiny` variable. The statement of the created task consist of the body of the method. Rule *return* adds the return value from a method call to the future identified by the task's `destiny` variable and time stamps the future at this time. Rule *get* describes how the **get** operation obtains the returned value.

The global advance of time is captured by the rule *tick*. This rule applies to global configurations in which all active statements are duration statements which have not reached their worst execution time or blocking get statements. These conditions are captured by the predicate `canAdvance` in Figure 5. When the *tick* rule is applicable, time may advance by any value t below the limit determined by the auxiliary *maximum time elapse* [14] function `mte`, which

```
crl [duration]: $\langle o,\ a,\ \{l \mid \textbf{duration}(b,w); s\} \circ q \rangle$
$\longrightarrow \langle o,\ a,\ \{l \mid s\} \circ q \rangle$ if $b \leq 0$ .

crl [async-call]: $\langle o,\ a, \{l \mid x{:=}e!m(\overline{e}); s\} \circ q \rangle$ clock($t$)
$\longrightarrow \langle o,\ a, \{l[x \mapsto n] \mid s\} \circ q \rangle\ \ m(t, [\![e]\!]^{none}_{aol}, [\![\overline{e}]\!]^{none}_{aol}, n)\ \ n$ if fresh($n$) .

crl [activation]: $\langle o,\ a, \{l \mid s\} \circ q \rangle\ \ m(t, o, \bar{v})$
$\longrightarrow \langle o,\ a, \{l \mid s\} \circ$ schedule(task($m(o,\bar{v})$),$q$) $\rangle$ .

crl [return]: $\langle o,\ a, \{l \mid$ return($e$);$s\} \circ q \rangle\ \ n$ clock($t$)
$\longrightarrow \langle o,\ a,\ \{l \mid s\} \circ q \rangle$ clock($t$) $\langle n, [\![e]\!]^{none}_{(a \circ l)}, t \rangle$ if $n = l$(destiny) .

crl [get]: $\langle o,\ a, \{l \mid x := e.\textbf{get}; s\} \circ q \rangle\ \ \langle n, v, t \rangle$
$\longrightarrow \langle o,\ a, \{l \mid x := v; s\} \circ q \rangle\ \ \langle n, v, t \rangle$ if $[\![e]\!]^{none}_{aol} = n$ .

crl [tick]: $\{C\} \longrightarrow \{\delta(C)\}$ in time $t$ if $t <$ mte($C$) $\wedge$ canAdvance($C$) .
```

Fig. 4. The semantics of Real-Time Creol in Real-Time Maude

```
op canAdvance: Configuration $\rightarrow$ Bool .
eq canAdvance(C1 C2) = canAdvance(C1) $\wedge$ canAdvance(C2) .
eq canAdvance($\langle o,\ a,\ \{l \mid \textbf{duration}(b,w); s\} \circ q\ \rangle$) = $w > 0$ .
eq canAdvance($\langle o,\ a, \{l \mid x := e.\textbf{get}; s\} \circ q\ \rangle\ \ n$) = true if $n = [\![x]\!]^{none}_{(\overline{a} \circ \overline{l})}$ .
eq canAdvance(C) = false [owise] .

op mte: Configuration $\rightarrow$ Time .
eq mte(C1 C2) = min(mte(C1), mte(C2)) .
eq mte($\langle o,\ a, \{l \mid \textbf{duration}(b,w); s\} \circ q\ \rangle$) = $w$ .
eq mte(C) = $\infty$ [owise] .

op $\delta_1$: Task Time $\rightarrow$ Task .
eq $\delta_1$({$l \mid s$},$t$) = {$l$[deadline $\mapsto l$(deadline) $- t$]|$s$} .

op $\delta_2$: TaskQueue Time $\rightarrow$ TaskQueue .
eq $\delta_2(\{l|s\} \circ q, t) = \delta_1(\{l|s\}, t) \circ \delta_2(q, t)$ .
eq $\delta_2(\epsilon, t) = \epsilon$

op $\delta_3$: Task Time $\rightarrow$ Task .
eq $\delta_3$({$l$|$\textbf{duration}(b,w); s$},$t$)
      = {$l$[deadline $\mapsto l$(deadline) $- t$] | $\textbf{duration}(b-t, w-t); s$} .
eq $\delta_3(\{l|s\}, t) = \{l$[deadline $\mapsto l$(deadline) $- t$]|$s\}$ [owise] .

op $\delta$: Configuration Time $\rightarrow$ Configuration .
eq $\delta$(C1 C2, $t$) = $\delta$(C1,$t$) $\delta$(C2,$t$) .
eq $\delta$(clock($t'$),$t$) = clock($t' + t$) .
eq $\delta$($\langle o,\ a,\ \{l|s\} \circ q\ \rangle$,$t$) = $\langle o,\ a,\ \delta_3(\{l|s\}) \circ \delta_2(q) \rangle$ .
eq $\delta$(C, $t$) = C [owise] .
```

Fig. 5. Definition of Auxiliary Functions

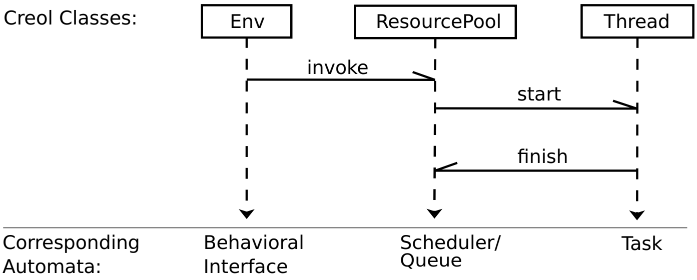


Fig. 6. Sequence diagram of a scenario from generation until completion of a task

finds the lowest remaining worst case execution time for any active task in any object in the configuration. Note that the blocking **get** operation allows time to pass arbitrarily while waiting for a return.

When time advances, the function δ, besides advancing the global clock, determines the effect of advancing time on the objects in the configuration, using the auxiliary functions δ_i, for $i = 1, 2, 3$, defined in Figure 5, to update the tasks. The function δ_1 decreases the deadline of a task. The function δ_2 applies δ_1 to all queued tasks; δ_2 has no effect on an empty queue ϵ. The function δ_3 additionally decreases the current best and worst case execution times of the active **duration** statements.

3 Schedulability Analysis

Schedulability analysis consists of checking whether tasks can be accomplished before their deadlines. For analysis, Real-Time Maude uses tick rules that advance time in discrete time steps, therefore verification of dense time models in Real-Time Maude is incomplete. Timed automata verification tools, e.g., UPPAAL, use symbolic time and thus cover the whole dense time space. In this section, we explain how to use timed automata for abstractly modeling concurrent objects and performing schedulability analysis. In this abstract modeling, infinite Creol programs are mapped to finite state automata.

We present a generalization of the automata-theoretic framework in [7] for schedulability analysis of concurrent objects. The overall real-time input/output behavior of an object is to be specified in an automaton called its *behavioral interface*. A behavioral interface abstracts from the detailed implementation of the object, which is in turn given in terms of its output behavior, given in the automata modeling the methods; and, the input enabled scheduler automaton that includes a queue to buffer the messages waiting to be executed.

In this paper we extend the schedulability analysis to an open system of concurrent objects. We explain this extension in terms of the thread pool example introduced in Section 2.1. Such a model can be synthesized from the sequence diagram in Figure 6 which depicts the life-cycle of a task from its generation until its completion. To allow communication between different automata, we

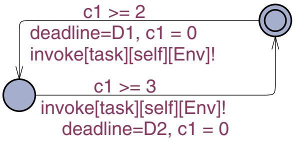


Fig. 7. Modeling a task generation pattern (right)

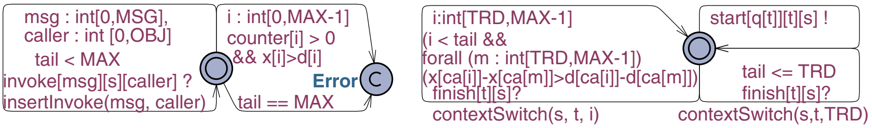
define a channel for each action in this diagram: Channel `invoke` has three parameters; namely, task name, the receiver and the sender. The parameters to channel `start` capture the task to be executed, the thread assigned to it, and the current object's identifier. Channel `finish` is parameterized in the identifiers of the executing thread and object. Next we discuss the different automata models corresponding to the three different life-lines in Figure 6.

Behavioral interfaces. The behavioral interface captures the overall real-time input/output behavior of an object while abstracting from its detailed implementation. Figure 7 shows a possible behavioral interface for our model of thread pools. This automaton is parameterized in the identifier of the thread pool, written `self`, and an identifier `Env` that represents any entity that may invoke a task on the thread pool. Since we only assume one task type in this example, we define a global constant `task` that will be used to identify this task.

We use a clock `c1` for modeling inter-arrival times and the global variable `deadline` is used for associating deadlines to each task generated. The tasks with different deadlines are interleaved and there is at least 2 and 3 time units of inter-arrival time between two consecutive task instances. This shows an example of non-uniform task generation pattern.

Scheduler and queue. The queue and the scheduling strategy are modeled in separate automata; together they represent the ResourcePool class. To model the ResourcePool, every thread is assumed to have a dedicated processing unit, but they share one task queue. We assume a fixed number `TRD` of threads given a priori. We separate the task queue in two parts: an *execution* part, consisting of the slots 0 to `TRD-1`, and a *buffer* part consisting of the rest of the queue. The execution part includes the tasks that are being executed. This part needs one slot for each thread and is therefore as big as the number of threads. The selection of a task from the buffer part to start its execution is based on a given scheduling strategies, e.g., EDF, FPS, etc.; in our example, we use EDF.

Figure 8(a) shows a queue of size `MAX` which stores the tasks in the order of their arrival; the queue is modeled by the array `q` and `tail` points to the first empty element in the queue. This automaton is parameterized in `s` which holds the identity of this object. This automaton can accept *any* task (whose identifier is between 0 and the constant `MSG`) by *any* caller (whose identifier is between 0 and the constant `OBJ`); this is seen as the UPPAAL 'select' statement over `msg` and `caller` on the `invoke` channel. This transition is enabled if the queue is not yet full (`tail < MAX`). To check for deadlines, a clock `x` is assigned to each



(a) A queue shared between threads (b) An EDF scheduler for each thread

Fig. 8. Allowing parallel threads to share a queue

task in the queue, which is reset when the task is added, i.e., in `insertInvoke` function. The queue goes to `Error` state if a task misses its deadline (`x[i] > d[i]`) or the queue is full (`tail == MAX`).

Figure 8(b) shows how a scheduling strategy can be implemented. This automaton should be replicated for every thread, thus parameterized in thread identity `t` as well as the object identity `s`. There will be one instance of this automaton for each slot `q[t]` in the execution part of the queue. This example models an EDF (earliest deadline first) scheduling strategy. The remaining time to the deadline of a task at position `i` in the queue is obtained by `x[ca[i]]-d[ca[i]]`. When the thread `t` finishes its current task (i.e. a synchronization on `finish[t][s]`), it selects the next task from the buffer part of the queue for execution by putting it in `q[t]`; this task is then immediately started (`start[q[t]][t][s]`).

To perform schedulability analysis by model checking, we need to find a reasonable queue length to make the model finite. The execution part of the queue is as big as the number of threads, and the buffer part is at least of size one. As in single-threaded situation of objects [7], a system is schedulable only if it does not put more than $\lceil D_{max}/B_{min} \rceil$ messages in its queue, where D_{max} is the biggest deadline in the system, and B_{min} is the best-case execution time of the shortest task. As a result, schedulability is equivalent to the `Error` state not being reachable with a queue of length $\lceil D_{max}/B_{min} \rceil$.

Tasks. A simple task model is given in Figure 9. In this model, the task has a computation time of between 5 to 6 time units. This corresponds to the model of the task given in the Creol code, which is defined in the start method of the Thread class and contains a skip statement followed by a duration. In general, a task model may be an arbitrarily complex timed automaton.

For schedulability analysis, one can experiment with different parameters. For example, one can choose different scheduling policies, like FCFS, EDF, etc. Since we assume that threads run in parallel, with more threads, we can handle more task instances (i.e., smaller inter-arrival times). Furthermore, if deadlines are too tight, schedulability cannot be guaranteed. Schedulability analysis amounts to checking whether the `Error` location in the queue automaton is reachable. Analysis shows that in the chosen settings, i.e., the selected inter-arrival times for the tasks and an EDF scheduler, this model cannot be schedulable with 2

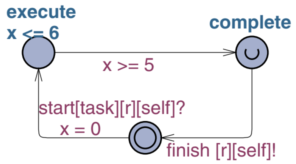


Fig. 9. Modeling a task

parallel threads, no matter how big the deadlines are. Intuitively, every 5 time units, two instances of the task may be inserted in the queue, and each task may take up to 6 time units to compute. With three parallel threads, these tasks can be scheduled correctly even with the deadline value of 6 time units for each task instance.

4 Conformance Testing

Our overall methodology for the schedulability analysis of a Real-Time Creol model consists of the following: We model the real-time pattern of incoming messages in terms of a timed automaton (the behavioral interface of the Creol model). Next we develop on the basis of sequence diagrams, which describe the observable behavior of the Creol model, automata abstractions of its overall real-time behavior. We analyze the schedulability of the product of this abstraction and the given behavioral interface (in for example Uppaal). Further, we define conformance between the Real-Time Creol model and its timed automaton abstraction with respect to the given behavioral interface in terms of inclusion of the timed traces of observable actions.

More specifically, let $\mathbf{C}$ denote a Creol model, i.e., a set of Creol classes, $\mathbf{B}$ a timed automaton specifying its behavioral interface and $\mathbf{A}$ a timed automata abstraction of the overall behavior of $\mathbf{C}$. We denote by $O(\mathbf{A} \parallel \mathbf{B})$ the set of timed traces of observable actions of the product of the timed automata $\mathbf{A}$ and $\mathbf{B}$. The set of timed traces of the timed automaton $\mathbf{B}$ we denote by $T(\mathbf{B})$. Further, given any timed trace $\theta \in T(\mathbf{B})$, the Creol class $\mathit{Tester}(\theta)$ denotes a Creol class which implements θ (see, for example, the class *Tester* in Figure 11). This class simply injects the messages at the times specified by θ. We denote by $O(\mathbf{C}, \mathit{Tester}(\theta))$ the set of timed traces of observable actions generated by the Real-Time Maude semantics of the Creol model $\mathbf{C}$ driven by θ. We now can define the conformance relation $\mathbf{C} \leq_{\mathbf{B}} \mathbf{A}$ by

$$O(\mathbf{C}, \mathit{Tester}(\theta)) \subseteq O(\mathbf{A} \parallel \mathbf{B}),$$

for every timed trace of observable actions $\theta \in T(\mathbf{B})$.

In this section we illustrate a method for testing conformance by searching for *counter-examples* to the conformance in terms of our running example. Note that a counter-example to the above conformance consist of a timed trace $\theta \in T(\mathbf{B})$ such that $O(\mathbf{C}, \mathit{Tester}(\theta)) \setminus O(\mathbf{A} \parallel \mathbf{B}) \neq \emptyset$.

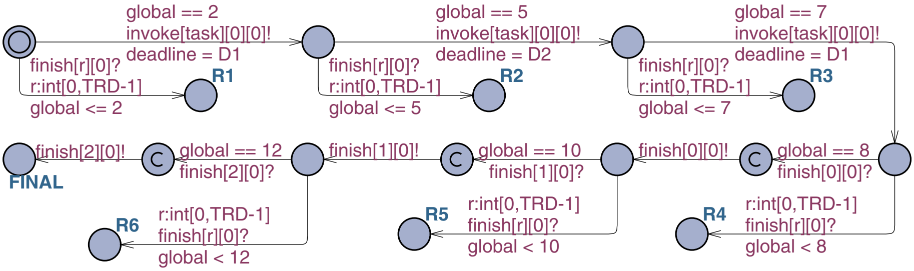


Fig. 10. Generating ready sets

4.1 Generating a Test Case

We first generate a timed trace $\theta = (t_1, a_1), \ldots, (t_n, a_n)$ by simulating the abstract timed automaton model **A** together with the behavioral interface **B**. To this end, we add a dummy automaton with a fresh clock `global` and an integer `time` which is incremented every time unit. This way we can find the absolute time interval in which every action in the trace has happened. In order to be able to search for a counter-example to conformance, we generate ready sets of observable actions generated by the behavioral abstraction of the Creol model. For each time interval between t_{i-1} and t_i in this trace and for every observable action a, we are interested in the following timed reachability property:

"`E<>` t_{i-1} `<= global && global <` t_i `&& a_f`",

where `a_f` denotes whether the observable action a has occurred in this interval. Instead of checking this property directly for every action, we encode it into one automaton as explained below (see Figure 10). This way, we avoid the need to add flags like `a_f` for every observable action and to go deep in the model to set it true when the corresponding action happens.

The algorithm to construct the automaton in Figure 10 for generating ready sets is as follows. Given a trace $\theta = (t_1, a_1), \ldots, (t_n, a_n)$, we first create a linear timed automaton T_θ with the locations $L = \{l_i \mid 1 \leq i \leq n+1\}$. By going from l_i to l_{i+1}, this automaton should ensure that action a_i happens at time $x == t_i$. This is done differently for inputs and outputs. Since the abstract Uppaal model **A** (i.e., excluding the behavioral interface) is input-enabled, the input actions only need to inject the task at the required time; namely with a transition from l_i to l_{i+1} with an `invoke` action. This transition should provide the required deadline. The output action `finish` is, however, produced by a task and consumed by the scheduler. To intercept this action, this automaton first mimics the scheduler by accepting the action, i.e., `finish?`, and then it mimics the task by issuing `finish!`.

We add to T_θ a location R_{ij} for each time interval between t_{i-1} and t_i and for each observable output action $o_j \in O(\mathbf{A} \parallel \mathbf{B})$, with one transition from l_i to

R_{ij} with a guard `global` $\leq t_i$ accepting the output action o_j; if a_i is the same as o_j, this transition is guarded by `global` $< t_i$. In our example, there is only one observable output action namely `finish`, but since a task can be taken by different threads, the `finish` action can be issued by different threads; therefore, the transitions for receiving this action should allow any thread identity `r` between 0 and `TRD-1`.

Finally, the reachability of the location R_{ij} implies that o_j must be included in the ready set R_i. We observe that in our example, only `R3` and `R4` are reachable; this is due to the possibility of finishing the first task instance in the interval $[7, 8]$. The consequent task instances can finish in the intervals $[10, 11]$ and $[12, 13]$, therefore, their completion does not contribute to an action in the ready sets. The test case including the ready sets and deadlines is:
(2, invoke(D1), {}) (5, invoke(D2), {}) (7, invoke(D1), {finish}) (8, finish, {finish}) (10, finish, {}) (12, finish, {})

4.2 Executing a Test Case in RT-Maude

Executing a test case amounts to injecting the inputs at the right times and looking for the right outputs at the right times. The system is input-enabled, so it accepts all the inputs. If the system under test cannot produce the expected output at the right time, the test fails. If along the test execution, the system under test can do an observable action that is neither the expected output nor in the ready-set, it is a counter-example to conformance. If the system can produce all expected outputs and no counter-example is found, the test passes in the sense that we are more confident that refinement holds and that the Creol model is schedulable. Notice that a counterexample to refinement does not necessarily imply non-schedulability in itself, but it shows an execution path that is likely to miss a deadline. We demonstrate this with the test-case from the previous subsection, repeated below:
(2, invoke(D1), {}) (5, invoke(D2), {}) (7, invoke(D1), {finish}) (8, finish, {finish}) (10, finish, {}) (12, finish, {})

We encode the input behavior given in the test-case as a complementary class that calls the methods of the model under test at the required times. For our running example, the code for the trace from previous subsection is given in Figure 11 (assuming $D1 = D2 = 6$).

By generating one instance of the ResourcePool class (with size 3 which gives us a schedulable Uppaal model, cf. Section 3) and one instance of the Tester class, we can check the output behavior of the Creol model against the test-case with consecutive search commands in Real-Time Maude as shown in Figure 12. In our case, the only observable output action is `finish`. To find a counter-example along this trace, we need to check whether a `finish` action can happen when it is not expected in the ready set, i.e., before time 2, between 2 and 5, between 8 and 10, or between 10 and 12. For each search command, we need to specify as time bound the duration since its start configuration, e.g., to search from C2 which is at time 5, we only need to search for another 2 time units to reach time 7.

```
class Tester (mut:ResourcePool){
  Void run(){
    duration(2,2);
    mut!invoke(6);
    duration(3,3); // 5-2 = 3
    mut!invoke(6);
    duration(2,2); // 7-5 = 2
    mut!invoke(6);
  }
}
```

Fig. 11. Tester Class

tsearch [1] {init} $\longrightarrow^*$ {Conf1 finish(T,M,E,N)} **in time** ≤ 2

If this search is successful, then we have a counter-example; otherwise, we continue with the following search:

tsearch [1] {init} $\longrightarrow^*$ {Conf1 invoke(2,M,E,N)} **in time** ≤ 2

If this search is not successful, then the test fails; otherwise, if Maude answers C1 → Conf1 then we continue with the following search:

tsearch [1] {C1 invoke(2,M,E,N)} $\longrightarrow^*$ {Conf2 finish(T,M,E,N)} **in time** ≤ 3

If this search is successful, then we have a counter-example; otherwise, we continue with the following search:

tsearch [1] {C1 invoke(2,M,E,N)} $\longrightarrow^*$ {Conf2 invoke(5,M,E,N)} **in time** ≤ 3

If this search is not successful, then the test fails; otherwise, if Maude answers C2 → Conf2 then we continue with the following search:

tsearch [1] {C2 invoke(5,M,E,N)} $\longrightarrow^*$ {Conf3 invoke(7,M,E,N)} **in time** ≤ 2

If this search is not successful, then the test fails; otherwise, if Maude answers C3 → Conf3 then we continue with the following search:

tsearch [1] {C3 invoke(7,M,E,N)} $\longrightarrow^*$ {Conf4 finish(8,M,E,N)} **in time** ≤ 1

If this search is not successful, then the test fails; otherwise, if Maude answers C4 → Conf4 then we continue with the following search:

tsearch [1] {C4 finish (8,M,E,N)} $\longrightarrow^*$ {Conf5 finish(T,M,E,N)} **in time** <2

If this search is successful, then we have a counter-example; otherwise, we continue with the following search:

tsearch [1] {C4 finish (8,M,E,N)} $\longrightarrow^*$ {Conf5 finish(10,M,E,N)} **in time** ≤ 2

If this search is not successful, then the test fails; otherwise, if Maude answers C5 → Conf5 then we continue with the following search:

tsearch [1] {C5 finish (10,M,E,N)} $\longrightarrow^*$ {Conf6 finish(T,M,E,N)} **in time** <2

If this search is successful, then we have a counter-example; otherwise, we continue with the following search:

tsearch [1] {C5 finish (10,M,E,N)} $\longrightarrow^*$ {Conf6 finish(12,M,E,N)} **in time** ≤ 2

Fig. 12. Executing the test-case for the thread-pools

It is possible to write a meta-level Maude script to automate the consecutive execution of these search commands, such that each search starts from the resulting configuration of the previous one. The technical details of how this can be done is beyond the scope of this paper.

5 Conclusion

We bridge the gap between automata theory and object orientation. We exploit the expressive power of Real-Time Maude to define the semantics of Real-Time Creol. We complement it with the analytical power of timed automata analysis tools like Uppaal. Based on this, we explained a methodology for schedulability analysis of open concurrent systems and applied it to the design and analysis of thread pools in an industrial communication platform. This methodology provides a separation of concerns between high-level modeling of architectural features of concurrent systems (in Creol) and their analysis for schedulability (using timed automata).

Behavioral interfaces are central to the analyses. Thread pools are analyzed for schedulability with respect to the task generation pattern given in the behavioral interfaces modeling the work-load. We also derive test cases from the behavioral interfaces for checking conformance between the timed automata abstractions and the Creol models, thus bridging the gap between the two levels of abstraction. We described a testing technique that is able to find counter-examples to conformance.

Future work consists, first of all, of an implementation of the method for testing conformance between a Creol model of a thread-pool and the timed automata models. Another line of future research consists of real-time extensions of the Creol language itself to support a full development cycle, so that one can generate code for application-specific schedulers from Creol models.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. The ASK community systems, http://www.ask-cs.com/
3. de Boer, F.S., Clarke, D., Johnsen, E.B.: A complete guide to the future. In: de Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 316–330. Springer, Heidelberg (2007)
4. de Boer, F.S., Grabe, I., Jaghoori, M.M., Stam, A., Yi, W.: Modeling and analysis of thread-pools in an industrial communication platform. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM 2009. LNCS, vol. 5885, pp. 367–386. Springer, Heidelberg (2009)
5. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: specification and programming in rewriting logic. Theoretical Computer Science 285(2), 187–243 (2002)

6. Closse, E., Poize, M., Pulou, J., Sifakis, J., Venter, P., Weil, D., Yovine, S.: TAXYS: A tool for the development and verification of real-time embedded systems. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 391–395. Springer, Heidelberg (2001)
7. Jaghoori, M.M., de Boer, F.S., Chothia, T., Sirjani, M.: Schedulability of asynchronous real-time concurrent objects. J. Logic and Alg. Prog. 78(5), 402–416 (2009)
8. Jaghoori, M.M., Longuet, D., de Boer, F.S., Chothia, T.: Schedulability and compatibility of real time asynchronous objects. In: Proc. Real Time Systems Symposium, pp. 70–79. IEEE CS, Los Alamitos (2008)
9. Johnsen, E.B., Owe, O.: An asynchronous communication model for distributed concurrent objects. Software and Systems Modeling 6(1), 35–58 (2007)
10. Kloukinas, C., Yovine, S.: Synthesis of safe, QoS extendible, application specific schedulers for heterogeneous real-time systems. In: Proc. 15th Euromicro Conference on Real-Time Systems (ECRTS 2003), pp. 287–294. IEEE Computer Society, Los Alamitos (2003)
11. Kyas, M., Johnsen, E.B.: A real-time extension of creol for modelling biomedical sensors. In: de Boer, F.S., Bonsangue, M.M., Madelaine, E. (eds.) FMCO 2008. LNCS, vol. 5751, pp. 42–60. Springer, Heidelberg (2009)
12. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. STTT 1(1-2), 134–152 (1997)
13. Meseguer, J.: Conditioned rewriting logic as a united model of concurrency. Theoretical Computer Science 96(1), 73–155 (1992)
14. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation 20(1-2), 161–196 (2007)
15. Schmaltz, J., Tretmans, J.: On conformance testing for timed systems. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 250–264. Springer, Heidelberg (2008)
16. Tretmans, J.: Model based testing with labelled transition systems. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) FORTEST. LNCS, vol. 4949, pp. 1–38. Springer, Heidelberg (2008)

Applications of Shellable Complexes to Distributed Computing
(Invited Talk)

Maurice Herlihy

Brown University
Computer Science Department

Introduction

This talk describes recent Joint work with Sergio Rajsbaum [3].

For models of concurrent computation in which processes may fail by crashing, each possible computation can be characterized as a *simplicial complex*, a geometric structure constructed by "gluing together" simplexes in a regular manner [6]. Informally, a complex is *k-connected* if it has no "holes" in dimension k or lower. It is known that if the complex corresponding to every such computation is k-connected, then one cannot solve $(k+1)$-set agreement [4,5,6].

A simplicial complex is *shellable* if it can be constructed by gluing a sequence of n-simplexes to one another along $(n-1)$-faces only. Shellable complexes have been studied in the combinatorial topology literature [1,2,7] because they have many nice combinatorial properties.

We can exploit these properties complexes to derive new and remarkably succinct tight (or nearly tight) lower bounds both on the connectivity of the associated complexes, and on solutions to the k-set agreement task in these models.

We consider a round-by-round model of computation, where we view each round as a map carrying simplexes to complexes. The principal insight is that if the single-round complex is shellable, then multi-round compositions preserve connectivity under certain easily-checkable conditions. These are theorems of combinatorial topology, independent of any model of computation.

We then show that for many classical models of computation, such as the synchronous, asynchronous, and semi-synchronous message-passing models, along with the asynchronous read-write memory model, each single-round complex is indeed shellable, so it becomes a straightforward exercise to derive tight (or nearly tight) bounds on when and if one can solve k-set agreement.

For asynchronous shared-memory models in which processes have access to "black-box" objects that solve consensus or k-set agreement, matters are a little more complicated. The single-round complex, while not itself shellable, is a simple union of shellable complexes, with a shellable nerve, and the same consequences follow.

Moreover, our results apply not just to the usual wait-free or t-resilient failure models, but to general *adversary* schedulers that can cause certain subsets of processes to fail, perhaps in a non-uniform way.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 19–20, 2010.

These results illustrate the power and continuing usefulness of topological methods for analyzing concurrent computation. Using a few well-known concepts from Combinatorial Topology, such as connectivity, nerves, and shellability, we can impose a common framework on a collection of heretofore unrelated models of computation, resulting in remarkably succinct proofs, not only of known results in each of these models, but also of new, previously-unknown results that extend classical wait-free and t-resilient bounds to general adversaries.

References

1. Björner, A.: Shellable and Cohen-Macaulay partially ordered sets. Transactions of the American Mathematical Society (1), 159–183 (July 1980)
2. Bruggesser, H., Mani, P.: Shellable decompositions of cells and spheres. Math. Scand. 29, 197–205 (1971)
3. Herlihy, M., Rajsbaum, S.: Concurrent computing and shellable complexes (submitted for Publication)
4. Herlihy, M., Rajsbaum, S.: Set consensus using arbitrary objects (preliminary version). In: PODC 1994: Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing, pp. 324–333. ACM, New York (1994)
5. Herlihy, M., Rajsbaum, S.: Algebraic spans. Mathematical Structures in Computer Science 10(4), 549–573 (2000)
6. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. J. ACM 46(6), 858–923 (1999)
7. Kozlov, D.: Combinatorial Algebraic Topology. Springer, Heidelberg (2007)

Concurrency and Composition in a Stochastic World

Christian Eisentraut[1], Holger Hermanns[1,2], and Lijun Zhang[3]

[1] Saarland University, Saarbrücken, Germany
[2] INRIA Grenoble, Rhône-Alpes, France
[3] DTU Informatics, Technical University of Denmark, Kgs. Lyngby, Denmark

Abstract. We discuss conceptional and foundational aspects of Markov automata [22]. We place this model in the context of continuous- and discrete-time Markov chains, probabilistic automata and interactive Markov chains, and provide insight into the parallel execution of such models. We further give a detailled account of the concept of relations on distributions, and discuss how this can generalise known notions of weak simulation and bisimulation, such as to fuse sequences of internal transitions.

1 Introduction

Petri nets are a model of concurrency. Among the most successful and widespread variations we find a class of models tailored to performance and dependability evaluation, Generalised Stochastic Petri nets (GSPNs) [38,39]. GSPNs support stochastically timed behaviour and weighted immediate choices. A simple example GSPN is depicted in Figure 1. What we see are places and transitions, connected by directed arrows. There are two types of transitions, timed (drawn white) and immediate (drawn black) transitions. If enabled, the latter fire immediately, while the earlier fire after a delay that is distributed according to a negative exponential distribution. Immediate transitions have priority over timed transitions. Evaluation of a GSPN proceeds at the level of the reachability graph. That graph is reduced to a continuous-time Markov chain (CTMC), for which efficient steady-state and transient solvers are at hand [14,17]. Due to their formality, visual representation, and the availability of efficient evaluation support,

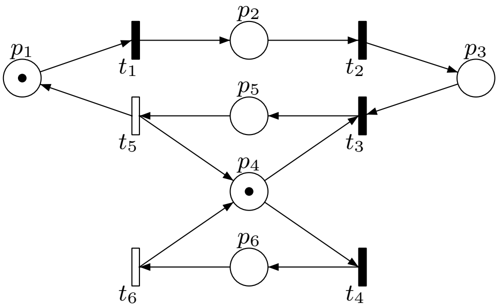


Fig. 1. A confused GSPN

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 21–39, 2010.

GSPNs have found widespread application in many diverse disciplines, including manufacturing, logistics, systems biology and so on [11,50,43,24].

Astonishingly, the above evaluation trajectory is incomplete. It is restricted to the class of *confusion-free* GSPNs. The net depicted in Figure 1 is confused. Confusion arises if a firing sequence admits the simultaneous enabling of multiple non-conflicting immediate transitions. Priorities and weights can eliminate confusion, but with strange effects. To shed some light on this phenomenon, assume that all weights of immediate transitions and also all rates of timed transitions are 1 for the above example. In this case, the steady state probability of a token being present in place p_5 is $4/11$. This probability changes to $2/5$ if one replaces the immediate transition t_2 and place p_2 by a direct arc from t_1 to p_3. In other words, adding or removing an immediate 'stutter step' somewhere in a weighted confused net may change the performance figures obtained for such nets. The issue of confusion has been discussed in the literature [13,15,37,6], but even after 25 years of pragmatic use, the final word on the foundational semantics of GSPNs is yet to be spoken.

Petri nets are visual, but not compositional. Process calculi provide compositional theories for complex systems, especially those involving communicating, concurrently executing components [8]. This paper is not about Petri nets with stochastically timed behaviour and weighted immediate choices. It is not about Petri nets at all. It is about compositional theories in a setting with stochastically timed behaviour and weighted immediate choices. The paper revolves around Markov automata [22], a model that indeed solves the semantic challenges hinted at above. This is achieved by harvesting and intertwining results established independently for two process calculi that each extend classical concurrency models in simple yet conservative fashions: Probabilistic automata [44,7,12] (PA), and interactive Markov chains [25] (IMC). Though different in flavour, both are equipped with compositional theories for strong and weak bisimilarities and corresponding equational theories.

In probabilistic automata, there is no global notion of time. Concurrent processes may perform random experiments inside a transition. This is represented by transitions of the form $s \xrightarrow{a} \mu$, where s is a state, a is an action label, and μ is a probability distribution over states. Labelled transition systems are instances of this model family, obtained by restricting to Dirac distributions (assigning full probability to single states). Thus, foundational concepts and results of standard concurrency theory are retained in their full beauty, and extend smoothly to the model of probabilistic automata. By restricting to Dirac distributions (assigning full probability to single states), labelled transition systems arise, and standard concurrency theoretical concepts and results are retained in their full beauty, and these extend smoothly to the model of probabilistic automata. In Markov automata, probabilistic automata are employed to give a genuine semantics to weighted immediate choices and their sequential or concurrent execution.

Interactive Markov chains in turn arise from classical concurrency models by adding a second type of transitions $s \overset{\lambda}{\twoheadrightarrow} s'$, that can embody random delays governed by a negative exponential distribution with some parameter λ. This twists the model to one that is running on a continuous time line, and where executions of actions take no time and happens immediately – unless an action can be blocked by the environment. This is linked to the process algebraic notion of maximal progress for internal actions. By dropping the second type of transitions, again, standard concurrency theory is regained in its entirety, and extends smoothly to the full IMC model. In Markov automata, IMC are used to represent stochastic timed behaviour and principal interaction possibilities.

Markov automata stand on the shoulders of PA and IMC. In a nutshell, the resulting model encompassed the expressiveness of GSPNs without semantic glitches, and with an entirely compositional theory. Due to the different time scales present in this model, this is a demanding endeavour. As in plain IMCs, internal immediate probabilistic transitions cannot be blocked and take no time to execute. Consequently, *MA* support fusing sequences of them. This implies that we need to partially ignore the branching structure of our probabilistic automata induced substructures when defining equalities, especially weak bisimilarity, on them. This is achieved by defining bisimulation as a relation on distributions over states, rather than as a relation on states, and by using the concept of distribution splitting, which is a concept of interest beyond the concrete Markov automata context. It allows for a surprisingly simple formulation of how transitions are fused, and allows to present various seemingly distinct preorders and equivalences in a unifying framework.

In this paper, we discuss conceptional and foundational aspects of Markov automata, partly rephrasing and complementing [22]. We place it in the context of some well-known and established models of concurrent computation that involve stochastically timed behaviour and weighted immediate choice, especially continuous- and discrete-time Markov chains, probabilistic automata and interactive Markov chains. We especially give insight into the parallel execution of such models, and discuss simulation and bisimulation relations on them. We show that the concept of relations on distributions generalises the respective standard relational notions. With this perspective on the different notions of bisimulation it becomes apparent how weak bisimulation for Markov automata appears as a natural generalisation of the constituent relations.

2 Preliminaries

(Sub-)distributions. A *subdistribution* μ over a set S is a function $\mu : S \mapsto [0,1]$ such that $\sum_{s \in S} \mu(s) \leq 1$. We denote by $Supp(\mu) = \{s \in S \mid \mu(s) > 0\}$ the support of μ and define the probability of $S' \subseteq S$ with respect to μ as $\mu(S') := \sum_{s \in S'} \mu(s)$. Let $|\mu| := \mu(S)$ denote the *size* of the subdistribution μ. We say μ is a *full distribution*, or distribution, if $|\mu| = 1$. Let $Dist(S)$ and $Subdist(S)$ be the set of distributions and subdistributions over S, respectively. For $s \in S$, we let $\Delta_s \in Dist(S)$ denote the *Dirac* distribution for s, i.e., $\Delta_s(s) = 1$. Let μ and μ' be two subdistributions. We define the subdistribution $\mu'' := \mu \oplus \mu'$ by $\mu''(s) = \mu(s) + \mu'(s)$, if $|\mu''| \leq 1$. Conversely, we say that μ'' can be split into μ and μ'. Or that (μ, μ') is a splitting of μ''. Moreover, if $x \cdot |\mu| \leq 1$, we let $x\mu$ denote the subdistribution defined by: $(x\mu)(s) = x \cdot \mu(s)$.

(Sub-)distributions can also be considered as sets over $S \times (0,1]$, where (s_1, r_1), $(s_2, r_2) \in S \times (0,1] \wedge s_1 = s_2$ implies $r_1 = r_2$, and where the second components of the elements sum up to a number smaller or equal to 1. The set view on subdistribution will be widely used throughout the paper. For example, to denote the distribution μ with $\mu(s_1) = 0.75$ and $\mu(s_2) = 0.25$, we may write $\mu = [\![(s_1, 0.75), (s_2, 0.25)]\!]$. Let for an element $s \in S$ and a subdistribution μ over S the expression $\mu{-}s$ denote the subdistribution that is obtained from μ by removing the pair $(s, \mu(s))$ from μ, if it exists. To make clear when we talk about sets representing subdistributions and when about general sets, we use $[\![$ and $]\!]$ for subdistributions, $\{$ and $\}$ for sets. Since $\oplus$ is associative and commutative, we may use the notation $\bigoplus_{i \in I}$ for arbitrary sums over a finite index set I.

Labelled trees. For $\sigma, \sigma' \in \mathbb{N}^*_{>0}$ we write $\sigma \leq \sigma'$ if there exists a (possibly empty) ϕ such that $\sigma\phi = \sigma'$. A partial function $\mathcal{T} : \mathbb{N}^*_{>0} \to L$, which satisfies

- if for $\sigma, \sigma' \in \mathbb{N}^*_{>0}$: $\sigma \leq \sigma'$ and $\sigma' \in \mathit{dom}(\mathcal{T})$ then $\sigma \in \mathit{dom}(\mathcal{T})$
- if $\sigma i \in \mathit{dom}(\mathcal{T})$ for $i > 1$, then also $\sigma(i-1) \in \mathit{dom}(\mathcal{T})$
- $\varepsilon \in \mathit{dom}(\mathcal{T})$

is called an *(infinite) L-labelled tree*. Let $\sigma \in \mathit{dom}(\mathcal{T})$: σ is called a leaf of $\mathcal{T}$ if there is no $\sigma' \in \mathit{dom}(\mathcal{T})$ such that $\sigma < \sigma'$. The empty word ε is called the root of $\mathcal{T}$. We denote the set of all leaves of $\mathcal{T}$ by $\mathit{Leaf}_{\mathcal{T}}$ and the set of all inner nodes by $\mathit{Inner}_{\mathcal{T}}$. If the tree has only one node, the root node, then this node is contained in both $\mathit{Inner}_{\mathcal{T}}$ and $\mathit{Leaf}_{\mathcal{T}}$. In any other case the two sets are disjoint. For a node σ of a tree $\mathcal{T}$ let $\mathit{Children}(\sigma) = \{\sigma i \mid \sigma i \in \mathit{dom}(\mathcal{T})\}$. In this paper, we consider L-labelled trees with finite branching, i.e., $|\mathit{Children}(\sigma)| < \infty$ for all node σ.

3 Markov Automata

We integrate probabilistic automata and interactive Markov chains into one model, defined by means of a twofold transition relation $\dashrightarrow$ and $\twoheadrightarrow$ [22]:

Definition 1. *A Markov automaton MA is a quintuple* $(S, \mathit{Act}, \dashrightarrow, \twoheadrightarrow, s_o)$*, where*

- *S is a nonempty finite set of states,*
- *Act is a set of actions containing the internal action τ,*
- *$\dashrightarrow \ \subset S \times \mathit{Act} \times \mathit{Dist}(S)$ is a set of immediate transitions, and*
- *$\twoheadrightarrow \ \subset S \times \mathbb{R}_{\geq 0} \times S$ is a set of timed transitions, and*
- *$s_o \in S$ is the initial state.*

We let s, u, v, t, E, F, G and their variants with indices range over S. For timed transitions, $\lambda, \mu \in \mathbb{R}_{\geq 0}$ denote rates of exponential distributions. For immediate transitions, a ranges over Act, and μ ranges over $\mathit{Dist}(S)$. A immediate transition $(E, a, \mu) \in \dashrightarrow$ is also denoted by $E \overset{a}{\dashrightarrow} \mu$, similarly we define $E \overset{\lambda}{\twoheadrightarrow} F$. We say an action $a \in \mathit{Act}$ is enabled in E, if there exists an immediate transition $E \overset{a}{\dashrightarrow} \mu$. A state $E \in S$ is called *stable* if τ is not enabled in E. If E is stable, we use the shorthand notation $E{\downarrow}$. We employ the *maximal progress assumption*. This means that if a state is not stable, time is not allowed to progress, making timed transitions out of this state irrelevant [28]. As in IMC, this assumption is not evident in the model, but is part of the equivalences defined on it.

We define a (nonnegative) real-valued function $\mathit{rate}_{\mathit{MA}} : S \times S \mapsto \mathbb{R}_{\geq 0}$, that calculates the rate to reach a state s' from a state s by

$$\mathit{rate}_{\mathit{MA}}(s, s') = \sum\{\lambda \mid s \overset{\lambda}{\twoheadrightarrow} s'\}.$$

Moreover, we define $\mathit{rate}_{MA}(s) := \sum_{s'} \mathit{rate}_{MA}(s, s')$ as the *exit rate* of s. The index is omitted if clear from the context. The delay associated with a state s that enables timed transitions is (negative) exponentially distributed with the exit rate $\mathit{rate}(s)$. In general, the probability to move from s to the successor state in s' equals the probability that the timed transitions that lead from s to $'$ *wins the race*. Therefore, the *discrete branching probability* to move to s' is given by $\mathbf{P}(s, s') := \frac{\mathit{rate}(s,s')}{\mathit{rate}(s)}$. For $s \in S$, we use $\mathbf{P}(s, \cdot)$ to denote this discrete branching distributions.

A *Dirac* distribution assigns full probability to a single outcome. We say that $\dashrightarrow$ is *Dirac* if the distributions occurring as third components of $\dashrightarrow$ are all Dirac, is *deterministic* if $E \overset{\alpha}{\dashrightarrow} \mu_1$ and $E \overset{\alpha}{\dashrightarrow} \mu_2$ implies that $\mu_1 = \mu_2$. Markov automata subsume many concurrent systems, which are discussed below.

1. *Labelled Transition Systems:* If $\Rrightarrow = \emptyset$ and $\dashrightarrow$ is Dirac, we obtain labelled transition systems.
2. *Discrete-time Markov chains:* If $\Rrightarrow = \emptyset$ and $|Act| = 1$ and $\dashrightarrow$ is deterministic, we obtain discrete-time Markov chains (DTMCs). In this case one usually ignores the single action, and writes it as a triple $(S, \mathbf{P}, s_0)$ where $\mathbf{P}$ is called the probability matrix, and is given by $\mathbf{P}(s, s') := \mu(s')$ provided $s \overset{\alpha}{\dashrightarrow} \mu$.
3. *Continuous-time Markov chains:* If $\dashrightarrow = \emptyset$ we obtain continuous-time Markov chains (CTMCs). It is commonly represented as a triple $(S, \mathbf{Q}, s_0)$ where $\mathbf{Q}$ is called the infinitesimal generator matrix, and is given by $\mathbf{Q}(s, s') := \mathit{rate}(s, s')$ provided $s \neq s'$ and $\mathbf{Q}(s, s) = \mathit{rate}(s, s) - \mathit{rate}(s)$. The latter reflects that in the original mathematical formulation of CTMCs it is impossible to make a difference between staying in state s, and jumping back to s from s.
4. *Probabilistic Automata:* If $\Rrightarrow = \emptyset$ we obtain probabilistic automata. If additionally $\dashrightarrow$ is deterministic, we arrive at Markov decision processes (MDPs).
5. *Interactive Markov chains:* If $\dashrightarrow$ is Dirac, we get interactive Markov chains (IMCs).

4 Parallel Composition

This section introduces parallel composition of *MA*s, and places it in the context of general operators for parallel composition, also motivating the rationale behind the semantic choices of the parallel operators in PA and IMC.

Assuming we are given two *MA*s $MA_1 = (S^1, Act^1, \dashrightarrow^1, \Rrightarrow^1, s_o^1)$ and $MA_2 = (S^2, Act^2, \dashrightarrow^2, \Rrightarrow^2, s_o^2)$, we consider a family of parallel operators $||_A$ indexed by some set $A \subseteq (Act^1 \cup Act^2) - \{\tau\}$. For a clear presentation we use these operators as syntactical means to denote some state $s_1 \; ||_{\mathsf{A}} \; s_2$, which arises by the parallel composition of s_1 and s_2. As syntactic sugar, we lift them to subdistributions as follows: for subdistributions $\mu_1 \in \mathit{Subdist}(S_1)$ and $\mu_2 \in \mathit{Subdist}(S_2)$, $\mu_1 \; ||_{\mathsf{A}} \; \mu_2$ denotes the subdistribution in $\mathit{Dist}(S_1 \times S_2)$ by distributing $||_{\mathsf{A}}$ element-wise. As an example, we have $(\mu_1 \; ||_{\mathsf{A}} \; \mu_2)(s_1 \; ||_{\mathsf{A}} \; s_2) := \mu_1(s_1) \cdot \mu_2(s_2)$.

Definition 2. *Let MA_1, MA_2 and A be as discussed above. The parallel operator can be applied to the two MAs to form the parallel composition $MA_1 \; ||_A \; MA_2 = (S, Act^1 \cup Act^2, \dashrightarrow, \Rrightarrow, s_o)$ of processes where*

- $S = \{s_1 \,||_A\, s_2 \mid (s_1, s_2) \in S^1 \times S^2\}$,
- $(s_1 \,||_A\, s_2, a, \mu_1 \,||_A\, \mu_2) \in \dashrightarrow$ *iff either*
  - $a \in A$ *and* $(s_i, a, \mu_i) \in \dashrightarrow^i$ *for each* $i \in \{1, 2\}$, *or*
  - $a \notin A$ *and* $(s_1, a, \mu_1) \in \dashrightarrow^1 \wedge \mu_2 = \Delta_{s_2}$ *or* $(s_2, a, \mu_2) \in \dashrightarrow^2 \wedge \mu_1 = \Delta_{s_1}$
- $(s_1 \,||_A\, s_2, \lambda, s_1' \,||_A\, s_2') \in \twoheadrightarrow$ *iff either*
  - *if for each* $i \in \{1, 2\}$, $\mathit{rate}_{MA_i}(s_i, s_i') > 0 \wedge s_i = s_i'$ *then* $\lambda = \mathit{rate}_{MA_1}(s_1, s_1') + \mathit{rate}_{MA_2}(s_2, s_2')$, *otherwise*
  - $\lambda = \mathit{rate}_{MA_1}(s_1, s_1')$ *and* $s_2' = s_2$, *or* $\lambda = \mathit{rate}_{MA_2}(s_2, s_2')$ *and* $s_1' = s_1$,
- $s_o = s_o^1 \,||_\mathsf{A}\, s_o^2 \in S^1 \times S^2$ *is the initial state.*

In a process algebraic setting, the style of defining this operator can be made more elegant [34], but this is not the topic of this paper.

4.1 The Roots of *MA* Parallel Composition

It is illustrative to relate this operator to the ones it is derived from.

1. Whenever MA_1 and MA_2 are *labelled transitions systems*, $MA_1 \,||_A\, MA_2$ reduces to LOTOS-style parallel composition [9].
2. Whenever MA_1 and MA_2 are *discrete-time Markov chains* over the same singleton set Act, $MA_1 \,||_{Act}\, MA_2$ reduces to the synchronous product of the models, where both Markov chains proceed in lockstep.
3. Whenever MA_1 and MA_2 are *continuous-time Markov chains*, $MA_1 \,||_\emptyset\, MA_2$ reduces to the independent and concurrent execution of the two Markov chains, justified by the memoryless property.
4. Whenever MA_1 and MA_2 are *probabilistic automata*, $MA_1 \,||_A\, MA_2$ agrees with the essence of the parallel composition for PA [45] (neglecting minor differences in synchronisation set constructions).
5. Whenever MA_1 and MA_2 are *interactive Markov chains*, $MA_1 \,||_A\, MA_2$ reduces to the parallel composition for IMC [25].

A few further remarks are in order (despite they may seem obvious to many readers): *MA* takes the LOTOS-style parallel operator $||_A$ as a basis, but we could have equally well opted for CCS style [40], CSP style [30], asynchronous or synchronous π-calculus style [41], I/O style [36], etc. From a pragmatic perspective, the $||_A$-operator is a convenient 'Swiss army knife'. It can, as special cases, express asynchronous interleaving ($||_\emptyset$), synchronous product ($||_{Act}$ with $|Act| = 1$). It can also be used to encode shared variable communication, as well as asynchronous message passing communication. Shared variables can be modelled as separate *MA*, where states correspond to variable valuations, and transitions are put in place for reading and updating the state. Similarly, asynchronous message passing channels can be encoded as *MA* that keep memory of the messages in transit (see e.g. [4, Chapter 2] for details).

We mention this to make clear that a properly and well understood semantics for this one operator is the nucleus for a well understood semantics of essentially any prevailing communication paradigms found in the real world. Since the models developed with *MA* (just like GSPN, IMC, PA) are meant to be designed and evaluated in order to provide insight into performance and dependability of the system being modelled, a well understood semantics is essential.

4.2 A Connection between DTMCs and CTMCs

To shed some more light on the parallel behaviour of these models, we state an interesting observation relating the interleaving semantics for CTMCs to the synchronous semantic for DTMCs. They are both derived from our single parallel composition for *MA*s, but why do they make sense after all? This section establishes a probably astonishing connection between the two.

Geometric and Exponential Distributions. It is well known – and we thus here take it for granted – that for any state in a CTMC, the sojourn time of that state is exponentially distributed, and likewise, that for any state in a DTMC, the sojourn time of that state is geometrically distributed.

Furthermore, the exponential distribution can be considered as the limit of the geometric distribution, in the following sense: Let continuous random variable X be exponentially distributed with parameter $\lambda > 0$, i.e. its density is $f_X(t) = \lambda e^{-\lambda t}$. Further, for $\Delta > 0$ with $p = \lambda\Delta < 1$, we consider the geometrically distributed discrete random variable $X_{\lambda\Delta}$ with parameter $\lambda\Delta$, i.e., with density function $f_{X_{\lambda\Delta}}(k) = \lambda\Delta(1 - \lambda\Delta)^{k-1}$. Then the definition of Euler's number implies that the density of X at time point $t > 0$ can be approximated by the density function of $X_{\lambda\Delta}$ at step $\lceil \frac{t}{\Delta} \rceil$, formally:

$$\forall t > 0.\ f_X(t) = \lim_{\Delta \to 0} f_{X_{\lambda\Delta}}(\lceil t/\Delta \rceil)/\Delta$$

One may consider Δ as a step size in continuous time (used to perform a single Bernoulli experiment, from which the geometric distribution is derived). This view is helpful in the discussion that follows.

Discretised Markov Chain. Let $M = (S, \mathbf{Q}, s_0)$ be a CTMC, and Δ be a sufficiently small step size. In the classical terminology [33], this CTMC is a family $\{C(t)\}$ of random variables each taking values in S, indexed by (continuous) time t, that obeys the Markov property. From this, we can derive a *Δ-discretised* DTMC $M_\Delta = (S, \mathbf{P}_\Delta, s_0)$ by: $\mathbf{P}_\Delta(s, s') = \mathit{Prob}(C(\Delta) = s' \mid C(0) = s)$. It holds that:

- $\mathbf{P}_\Delta(s, s')$ equals $\mathbf{Q}(s, s')\Delta + o(\Delta)$, provided $s \neq s'$, and otherwise
- $\mathbf{P}_\Delta(s, s)$ equals $1 + \mathbf{Q}(s, s)\Delta + o(\Delta)$.

Here, $o(\Delta)$ subsumes the probabilities to pass through intermediate states between s and s' during the interval Δ, and we have $\mathbf{P}_\Delta(s, s) \in (0, 1)$ – for sufficiently small Δ.

Moreover, the rate between s and s' can be derived from the derivative:

- $\mathbf{Q}(s, s') = \lim_{\Delta \to 0} \mathbf{P}_\Delta(s, s')/\Delta$, provided $s \neq s'$, and otherwise
- $\mathbf{Q}(s, s) = \lim_{\Delta \to 0}(-\sum_{s' \neq s} \mathbf{P}_\Delta(s, s')/\Delta)$.

This observation justifies that the behaviour of a CTMC can be approximated arbitrarily closely by a Δ-discretised DTMC, just by choosing Δ small enough, since in the limit $\mathbf{Q} = \lim_{\Delta \to 0}(\mathbf{P}_\Delta - \mathbf{I})/\Delta$, where $\mathbf{I}$ denotes the identity matrix. The limit is understood element-wise. All the above facts can, albeit usually stated in a different flavour, be found in many textbooks on Markov chains, for example in [47].

What is not found in textbooks is the question whether this approximation is compatible with parallel composition: For two CTMCs M and M', let M_Δ and M'_Δ denote the corresponding Δ-discretised DTMCs respectively, which we assume labelled over the same singleton set Act. We now consider the synchronous product $M_\Delta \;||_{Act}\; M'_\Delta$, where the two Markov chains evolve in lockstep with the step size – on a continuous time line – being Δ. Now, how does this product relate to $M \;||_\emptyset\; M'$, the parallel composition of the CTMCs M and M' under interleaving semantics? The following lemma answers this.

Lemma 1. *Let $M = (S, \mathbf{Q}, s_0)$ and $M' = (S', \mathbf{Q}', s'_0)$ be two CTMCs, let M_Δ, M'_Δ denote the probability matrices in the discretised DTMCs, respectively. Moreover, let $M \;||_\emptyset\; M' = (S \;||_\emptyset\; S', \mathbf{Q}^{||}, s_0)$ and $M_\Delta \;||_{Act}\; M'_\Delta = (S \;||_{Act}\; S', \mathbf{P}^{||}_\Delta, s_0)$. Then,*

$$\mathbf{Q}^{||} = \lim_{\Delta \to 0} (\mathbf{P}^{||}_\Delta - \mathbf{I})/\Delta$$

Proof. Let $s, t \in S$ and $s', t' \in S'$. We consider a few cases:

1. Assume $s \neq t$ and $s' \neq t'$. By the synchronised product in the DTMCs, we have:
$$\mathbf{P}^{||}_\Delta(s \;||_{Act}\; s', t \;||_{Act}\; t') = \mathbf{P}_\Delta(s,t) \cdot \mathbf{P}'_\Delta(s',t') = \mathbf{Q}(s,t)\mathbf{Q}'(s',t')\Delta^2 + o(\Delta)$$
It holds now $\lim_{\Delta\to 0} \mathbf{P}^{||}_\Delta(s \;||_{Act}\; s', t \;||_{Act}\; t')/\Delta = 0$. By definition of $||_\emptyset$ we also have $\mathbf{Q}^{||}(s \;||_\emptyset\; s', t \;||_\emptyset\; t') = 0$.
2. Now we consider the case $s = t$ and $s' \neq t'$. Under this assumption we have that $\mathbf{Q}^{||}(s \;||_\emptyset\; s', t \;||_\emptyset\; t') = \mathbf{Q}'(s', t')$, and moreover, $\mathbf{P}_\Delta(s,s) = 1 + \mathbf{Q}(s,s)\Delta + o(\Delta)$. The rest can be shown similarly as previous case.
3. Finally we consider the case $s = t$ and $s' = t'$. In this case we have
$$\begin{aligned}\mathbf{P}^{||}_\Delta(s \;||_{Act}\; s', s \;||_{Act}\; s') &= (1 + \mathbf{Q}(s,s)\Delta + o(\Delta)) \cdot (1 + \mathbf{Q}'(s',s')\Delta + o(\Delta)) \\ &= 1 + (\mathbf{Q}(s,s) + \mathbf{Q}'(s',s'))\Delta + o(\Delta)\end{aligned}$$
Thus: $\lim_{\Delta\to 0}(\mathbf{P}^{||}_\Delta(s \;||_{Act}\; s', s \;||_{Act}\; s') - 1)/\Delta = \mathbf{Q}(s,s) + \mathbf{Q}'(s',s')$, which is exactly $\mathbf{Q}^{||}(s \;||_\emptyset\; s', s \;||_\emptyset\; s')$.

The above lemma derives the interleaving semantics for CTMCs through the limiting behaviour of their discretised DTMCs evolving synchronously.

5 Simulations and Bisimulations

We now discuss equivalences and preorders for *MA* and submodels thereof. We first introduce a notation that makes our further discussion more compact, at the price of mildly reduced readability. It enables a uniform treatment of immediate and timed transitions. In doing so, we introduce the special action $\chi(r)$ to denote the exit r rate of a state. Moreover, we let $Act^\chi := Act \cup \{\chi(r) \mid r \in \mathbb{R}_{\geq 0}\}$, and $\alpha, \beta, ...$ range over this set.

Definition 3. *Let MA $= (S, Act, \dashrightarrow, \twoheadrightarrow, s_o)$ be an MA. Let $E \in S$ and $\alpha \in Act^\chi$. We write $E \xrightarrow{\alpha} \mu$ if*

- *$E \overset{\alpha}{\dashrightarrow} \mu \wedge \alpha \in Act$ or*
- *$E{\downarrow} \wedge r = \mathit{rate}(E) \wedge \alpha = \chi(r) \wedge \mu = \mathbf{P}(E, \cdot)$.*

5.1 Strong Simulations and Bisimulation

Strong Simulations. Strong simulations for DTMCs were originally introduced [31] using the concept of *weight functions*. Weight function have since become a standard way of formalising specification and refinement relations between probabilistic processes. Given distributions μ, μ' and a relation R, a weight function δ requires assigning weights in $[0, 1]$ for every pair of states in R, such that $\mu(s) = \sum_{s' \in S} \delta(s, s')$ and symmetrically $\mu'(s') = \sum_{s \in S} \delta(s, s')$. Owed to the need of assigning these weights, many proofs need human ingenuity and become certainly nontrivial. This is felt, for instance, in the proof of transitivity of strong similarity. As a consequence, a few equivalent reformulations of weight functions have emerged. One of them is based on computing the *maximum flow* on a particular network constructed out of μ, μ' and R [2]. This has been successfully exploited in decision procedures for various simulation preorders [2,52,21]. Below, we use another, rather recent, reformulation [51,21]: the existence of a weight function for distributions μ, μ' with respect to R is equivalent to the statement that, $\mu(A) \leq \mu(R(A))$ for every set A of states. Here, we write $R(A) := \{s' \mid (s, s') \in R \ \wedge \ s \in A\}$. This alternative characterisation of weight function provides a very intuitive interpretation: for every set A, μ' assigns a higher probability to the related set $R(A)$ relative to μ. The inspiration for this definition stems from [19], in which strong simulation relations are, in addition, required to be preorders.

Definition 4 (Strong Simulations). *Let MA* $= (S, Act, \dashrightarrow, \twoheadrightarrow, s_o)$ *be an MA. Let* $\mathcal{R}$ *be a binary relation on* S*. Then,* $\mathcal{R}$ *is a strong simulation iff* $E\mathcal{R}F$ *implies:*

1. *for all* $\alpha \in Act$*:* $E \xrightarrow{\alpha} \mu$ *implies* $F \xrightarrow{\alpha} \mu'$ *for some distribution* μ' *such that* $\mu(A) \leq \mu'(R(A))$ *for all* $A \subseteq S$*, and*
2. *for all* $r \in \mathbb{R}_{\geq 0}$: $E \xrightarrow{\chi(r)} \mu$ *implies* $F \xrightarrow{\chi(r')} \mu'$ *for some distribution* μ' *and* $r' \in \mathbb{R}_{\geq 0}$ *such that* $r \leq r'$ *and* $\mu(A) \leq \mu'(R(A))$ *for all* $A \subseteq S$*.*

State F strongly simulates E, written $E \precsim F$, if (E, F) is contained in some strong simulation.

1. On *labelled transitions systems* $\precsim$ coincides with standard strong similarity [42,1].
2. On *discrete-time Markov chains* $\precsim$ coincides with strong similarity [31].
3. On *continuous-time Markov chains* $\precsim$ coincides with strong similarity [5].
4. On *probabilistic automata* $\precsim$ coincides with strong similarity [46].
5. On *interactive Markov chains* $\precsim$ coincides with strong similarity [32] if strengthening $r \leq r'$ to $r = r'$.

Strong Bisimulations. Strong bisimilarity in its diverse flavours is the most prominent equivalence relation for probabilistic models. For *MA*, the obvious combination of strong bisimilarity for IMC and strong bisimilarity for PA can be phrased as follows:

Definition 5. *Let MA* $= (S, Act, \dashrightarrow, \twoheadrightarrow, s_o)$ *be an MA. Let* $\mathcal{R}$ *be an equivalence relation on* S*. Then,* $\mathcal{R}$ *is a strong bisimulation iff* $E\mathcal{R}F$ *implies for all* $\alpha \in Act^{\chi}$*:* $E \xrightarrow{\alpha} \mu$ *implies* $F \xrightarrow{\alpha} \mu'$ *with* $\mu(C) = \mu'(C)$ *for all* $C \in S/\mathcal{R}$*.*

Two states E and F are strongly bisimilar, written $E \sim F$, if (E, F) is contained in some strong bisimulation.

1. On *labelled transitions systems* $\sim$ is strong bisimilarity [42,40].
2. On *discrete-time Markov chains* $\sim$ coincides with strong bisimilarity [35] and corresponds to lumpability [33].
3. On *continuous-time Markov chains* $\sim$ coincides with lumping equivalence [26].
4. On *probabilistic automata* $\sim$ coincides with strong bisimilarity [46].
5. On *interactive Markov chains* $\sim$ coincides with strong bisimilarity [25].

For PA, coarser relations than strong bisimilarity and strong similarity – still treating internal transitions as ordinary transitions – are established based on the concept of combined transitions. The resulting relations are called *strong probabilistic (bi-)similarities* [46,45]. They can also be defined directly in our setting, by replacing $F \xrightarrow{a} \mu'$ by a convex combination of several a-labelled transitions starting in F. Details are left to the interested reader.

5.2 Weak Transitions

Weak transitions for probabilistic systems have been defined in the literature via probabilistic executions in [44], trees [20], or infinite sums [18]. We adopt the tree notation here. The material presented below concerning weak transitions provides no innovation over the classical treatment, it is included for the benefit of the reader.

We consider in the following $S \times \mathbb{R}_{\geq 0} \times Act^{\chi} \cup \{\epsilon\}$-labelled trees. Briefly, a node in such trees is labelled by the corresponding state, probability of reaching this node, and the chosen action (including the special action for timed transitions) to proceed. For a node σ we write $\mathit{Sta}_t(\sigma)$ for the first component of $t(\sigma)$, $\mathit{Prob}_t(\sigma)$ for the second component of $t(\sigma)$ and $\mathit{Act}_t(\sigma)$ for the third component of $t(\sigma)$.

Definition 6. *Let MA* $= (S, Act, \dashrightarrow, \twoheadrightarrow, s_o)$ *be an MA. A* (weak) transition tree $\mathcal{T}$ *is a* $S \times \mathbb{R}_{\geq 0} \times Act^{\chi} \cup \{\epsilon\}$*-labelled tree that satisfies the following condition:*

1. $0 < \mathit{Prob}_{\mathcal{T}}(\varepsilon) \leq 1$
2. $\forall \sigma \in \mathit{Leaf}(\mathcal{T}) : \mathit{Act}_{\mathcal{T}}(\sigma) = \epsilon.$
3. $\forall \sigma \in \mathit{Inner}(\mathcal{T}) \setminus \mathit{Leaf}(\mathcal{T}) : \exists \mu : \mathit{Sta}_{\mathcal{T}}(\sigma) \xrightarrow{\mathit{Act}_{\mathcal{T}}(\sigma)} \mu$ *and*

$$\mathit{Prob}_{\mathcal{T}}(\sigma) \cdot \mu = [\![(\mathit{Sta}_{\mathcal{T}}(\sigma'), \mathit{Prob}_{\mathcal{T}}(\sigma')) \mid \sigma' \in \mathit{Children}_{\mathcal{T}}(\sigma)]\!]$$

4. $\sum_{\sigma \in \mathit{Leaf}} \mathit{Prob}_{\mathcal{T}}(\sigma) = \mathit{Prob}_{\mathcal{T}}(\varepsilon).$

We call the tree weak, if $\mathit{Prob}_{\mathcal{T}}(\varepsilon) < 1.$

Restricting Act^{χ} to Act, a transition tree $\mathcal{T}$ corresponds to a probabilistic execution fragment: it starts from $\mathit{Sta}_{\mathcal{T}}(\epsilon)$, and resolves the non-deterministic choice by executing the action $\mathit{Act}_{\mathcal{T}}(\sigma)$ at the inner node σ. The second label of σ is then the probability of reaching $\mathit{Sta}_{\mathcal{T}}(\sigma)$, starting from $\mathit{Sta}_{\mathcal{T}}(\epsilon)$ and following the selected actions. If in a node σ the timed transition is chosen, the third label $\mathit{Act}_{\mathcal{T}}(\sigma) \in \mathbb{R}_{\geq 0}$ represents the exit rate of $\mathit{Sta}_{\mathcal{T}}(\sigma)$. In this case, a child σ' is reached with $\mathit{Prob}_{\mathcal{T}}(\sigma)$ times the discrete branching probability $\mathbf{P}(\mathit{Sta}_{\mathcal{T}}(\sigma), \mathit{Sta}_{\mathcal{T}}(\sigma'))$.

An *internal transition tree* $\mathcal{T}$ is a transition tree where each $\mathit{Act}_{\mathcal{T}}(\sigma)$ is either τ or ϵ. Let $\mathcal{T}$ be a transition tree. Then the subdistribution associated with $\mathcal{T}$, denoted by $\mu_{\mathcal{T}}$, is defined as

$$\mu_{\mathcal{T}} \overset{\text{def}}{=} \bigoplus_{\sigma \in \mathit{Leaf}_{\mathcal{T}}} [\![(\mathit{Sta}_{\mathcal{T}}(\sigma), \mathit{Prob}_{\mathcal{T}}(\sigma))]\!] \,.$$

We say subdistribution $\mu_{\mathcal{T}}$ is *induced* by $\mathcal{T}$. Obviously, $\mu_{\mathcal{T}}$ is a full distribution if we have $\mathit{Prob}_{\mathcal{T}}(\varepsilon) = 1$. With the above definitions we are now able to express weak transitions:

Definition 7. *For $E \in S$ and μ a full distribution we write*

- *$E \Longrightarrow \mu$ if μ is induced by some internal transition tree $\mathcal{T}$ with $\mathit{Sta}_{\mathcal{T}}(\varepsilon) = E$.*
- *$E \overset{\alpha}{\Longrightarrow} \mu$ if μ is induced by some transition tree $\mathcal{T}$ with $\mathit{Sta}_{\mathcal{T}}(\varepsilon) = E$, where on every maximal path from the root at least one node σ is labelled $\mathit{Act}_{\mathcal{T}}(\sigma) = \alpha$. In case that $\alpha \neq \tau$, then there must be exactly one such node on every maximal path. And all other inner nodes must be labelled by τ.*
- *$E \overset{\hat{\alpha}}{\Longrightarrow} \mu$ if $\alpha = \tau$ and $E \Longrightarrow \mu$ or $E \overset{\alpha}{\Longrightarrow} \mu$.*

For all three transition relations we say that the transition tree that induces μ also induces the transition *to μ.*

Note that $E \Longrightarrow \Delta_E$ and $E \overset{\hat{\tau}}{\Longrightarrow} \Delta_E$ holds independently of the actual transitions E can perform, whereas $E \overset{\tau}{\Longrightarrow} \Delta_E$ only holds if $E \overset{\tau}{\dashrightarrow} \Delta_E$. For all $\alpha \neq \tau$, $E \overset{\hat{\alpha}}{\Longrightarrow} \mu$ is identical to $E \overset{\alpha}{\Longrightarrow} \mu$. Below we define the notion of *combined transitions* [44], which arise as convex combination of a set of transitions with the same label, including the label representing timed transitions.

Definition 8. *We write $E \overset{\alpha}{\Longrightarrow}_C \mu$, if $\alpha \in \mathit{Act}^{\chi}$ and there is a finite indexed set $\{(c_i, \mu_i)\}_{i \in I}$ of pairs of positive real valued weights and distributions such that $E \overset{\alpha}{\Longrightarrow} \mu_i$ for each $i \in I$ and $\sum_{i \in I} c_i = 1$ and $\mu = \bigoplus_{i \in I} c_i \mu_i$.*

We say that $E \overset{\alpha}{\Longrightarrow}_C \mu$ is *justified* by the set $\{(c_i, \mu_i)\}_{i \in I}$. Transitions relations from states to distributions can be generalised to take (sub)distributions μ to (sub)distributions, by weighting the result distribution of the transition of each element $E \in \mathit{Supp}(\mu)$ by $\mu(E)$.

Definition 9. *Let $\rightsquigarrow \in \left\{ \overset{\hat{}}{\Longrightarrow}, \Longrightarrow, \dashrightarrow, \twoheadrightarrow, \longrightarrow \right\}$. Then, we write $\mu \rightsquigarrow \gamma$ if $\gamma = \bigoplus_{s_i \in \mathit{Supp}(\mu)} \mu(s_i)\mu_i$, where $s_i \rightsquigarrow \mu_i$ holds for all $s_i \in \mathit{Supp}(\mu)$.*

We say that $\mu \rightsquigarrow \gamma$ is *justified* by the transitions $s_i \rightsquigarrow \mu_i$.

5.3 Weak Simulations and Bisimulations over Subdistributions

Weak simulations and bisimulations are means to abstract from internal transitions and sequences thereof. In our setting this means that we intend to fuse distributions that arise from sequences of internal transitions in an *MA* into single, accumulated distributions.

Relating distributions. Bisimulations for PA and IMC have in the past been defined as relations on states, akin to bisimulations for LTS. The latter reside in a single state at every point in time during their execution, thus bisimulation relations on states are all that is needed to capture their behaviour adequately. In contrast, a stochastic system resides – for every point in time – in a certain state with a certain probability. The system behaviour is thus not correctly determined by the states occupied, but instead by probability distributions over states in which a system can reside at any point in time. It thus seems natural to define bisimulation relation as relations on distributions over states.

Several simulation relations for PA adopt this approach [46,44,48] in an asymmetric way, simulating a state by a distribution over states. Among these relations, probabilistic forward similarity [44] is the coarsest. We denote it by $\leq_{fwd}$. In Figure 2, state u is forward simulated by state v and vice versa. In an asymmetric way, $\leq_{fwd}$ achieves exactly what we intend to achieve for *MA*: we aim at fusing distribution along internal immediate transition sequences. It is however not obvious how to extend the definition of forward simulation, which relates states to distributions, to a *bisimulation* setting, which should then relate distributions to distributions. Even partially state-based approaches seem to fail, since in order to equate u and v, the two distributions $[\![(v', 1)]\!]$ and $[\![(E, \frac{1}{3}), (F, \frac{2}{3})]\!]$ must be considered equivalent. However, from a state-based point of view, the two distributions must be different, assuming that E and F represent states of different equivalence classes, since neither E nor F alone can then be equivalent to state v' .

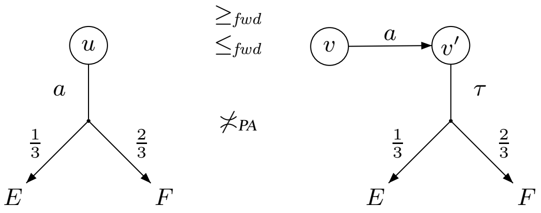


Fig. 2. Probabilistic forward simulation versus probabilistic weak bisimulation

We instead advocate a bisimulation-over-distribution approach to define a notion of weak bisimilarity that satisfies our demands [22]. In the sequel, we reiterate and rephrase several interesting aspects of this approach, and characterise the semantical relationship between weak bisimilarity and the standard bisimilarities of PA and IMC.

Naïve Weak Bisimulation over States. In the following, we will show that the standard bisimulations of PA and IMC can easily by cast as relations over distributions. In favour of a concise presentation, we will not consider the standard bisimulations for PAs and IMC separately, but only investigate a relation $\asymp$, which we define as a direct combination of IMC and PA weak bisimilarity, such that on the IMC submodel we obtain IMC weak bisimilarity [25], and on the PA submodel, we obtain (a divergence sensitive variation) of PA weak bisimilarity, namely stable weak bisimilarity. The variation in the latter case is owed to the maximal progress assumption, inherited from IMC and necessary for general *MA*s. This has, however, no influence on the technical development.

We have first introduced this relation in [22, Def. 10], as a naïve approach of defining a suitable weak equivalence for *MA*s. As we have argued there, this relation is, however, not suitable to achieve the intended effect of fusing internal transitions. We thus call this relation *naïve weak bisimulation*.

Definition 10 (Naïve Weak Bisimulation). *Let* $MA = (S, Act, \dashrightarrow, \twoheadrightarrow, s_o)$ *be a* *MA. For two states* $s, s' \in S$, $s \asymp s'$ *holds if* $(s, s') \in \mathcal{E}$ *for some equivalence relation* $\mathcal{E}$ *on* S *for which* $E\mathcal{E}F$ *implies for all* $\alpha \in Act^{\chi}$ *and for all equivalence classes* C *of* $\mathcal{E}$, $E \xrightarrow{\alpha} \mu$ *implies* $F \overset{\hat{\alpha}}{\Longrightarrow}_C \gamma$ *for some* γ *and* $\forall C \in S/\mathcal{E} : \mu(C) = \gamma(C)$.

Weak (Bi-)simulations over Subdistributions. We will now introduce two weak bisimulation – and also two weak simulation – relations that relate distributions (or subdistributions, to be precise). One of them is *weak bisimulation* for Markov automata as introduced in [22, Def. 11], the other bisimulation appears new. We will show that this new bisimulation relation on distributions induces a bisimilarity on states that coincides with naïve weak bisimulation, which itself is defined directly over states. This strong connection bridges between the state-based and distribution-based approach and allows us to make precise their relationship. We call the new relation *semi-weak* bisimulation, since it is *weak*, meaning partially oblivious to internal transitions, but yet finer than weak bisimulation for Markov automata.

Both relations rely on the idea of equating subdistributions exactly when they can be split into subdistributions, such that for each component in one splitting there exists a related component of the other splitting that exhibits identical behaviour, and vice versa. Remarkably, the definitions only differ in one specific point. For *semi-weak* (bi)simulation, splittings need to be immediately related to match their behaviour. For *weak* (bi)simulation, we relax the conditions such that it suffices if equated distributions are able to reach distributions that can then be split suitably by internal transitions. To make explicit that the relations only differ in the way subdistributions are split, we define two sets of possible splittings of a subdistribution.

- For weak (bi)simulation, we use a set
$$\mathit{split}(\mu) = \{(\mu_1, \mu_2) \mid \exists \mu' : \mu \Longrightarrow_C \mu' \wedge \mu' = \mu_1 \oplus \mu_2\}$$
where each splitting of an internal successor subdistribution of μ is a valid splitting.
- For semi-weak (bi)simulation, we use a more restricted set
$$\mathit{split}^{\circ}(\mu) = \{(\mu_1, \mu_2) \mid \mu = \mu_1 \oplus \mu_2\}$$
where only direct splittings of μ are valid splittings.

Since weak and semi-weak bisimulation only differ in this one point, we will define them simultaneously in one definition. In what follows, the expression $\mathit{split}^{(\circ)}$ needs to be replaced by *split* in order to obtain weak bisimulation. Semi-weak bisimulation is obtained by replacing it by split°.

Definition 11 (Weak Bisimulations). *A relation* $\mathcal{R}$ *on subdistributions over* S *is called a* (semi-)weak bisimulation *iff whenever* $\mu_1 \mathcal{R} \mu_2$ *then for all* $\alpha \in Act^{\chi}$*:* $|\mu_1| = |\mu_2|$ *and*

1. *$\forall E \in \mathit{Supp}(\mu_1)$: $\exists {\mu_2}^g, {\mu_2}^s$: $({\mu_2}^g, {\mu_2}^s) \in \mathit{split}^{(\circ)}(\mu_2)$ and*
   (i) $[\![(E,\mu_1(E))]\!] \;\mathcal{R}\; {\mu_2}^g$ and $(\mu_1{-}E) \;\mathcal{R}\; {\mu_2}^s$
   (ii) whenever $E \xrightarrow{\alpha} \mu_1'$ for some μ_1' then ${\mu_2}^g \xRightarrow{\hat{\alpha}}_C \mu''$ and $(\mu_1(E) \cdot \mu_1') \;\mathcal{R}\; \mu''$
2. *$\forall F \in \mathit{Supp}(\mu_2)$: $\exists {\mu_1}^g, {\mu_1}^s$: $({\mu_1}^g, {\mu_1}^s) \in \mathit{split}^{(\circ)}(\mu_1)$ and*
   (i) ${\mu_1}^g \;\mathcal{R}\; [\![(F,\mu_1(F))]\!]$ and ${\mu_1}^s \;\mathcal{R}\; (\mu_2{-}F)$
   (ii) whenever $F \xrightarrow{\alpha} \mu_2'$ for some μ_2' then ${\mu_1}^g \xRightarrow{\hat{\alpha}}_C \mu''$ and $\mu'' \;\mathcal{R}\; (\mu_2(F) \cdot \mu_2')$

Two subdistributions μ and γ are (semi-)weak bisimilar, denoted by $\mu \approx^{(\circ)} \gamma$, if the pair (μ, γ) is contained in some (semi-)weak bisimulation.

It is worth noting that the weak bisimilarity $\approx$ in the above definition is identical to [22, Def. 11]. We immediately obtain simulation relations by removing Condition 2.

Definition 12 (Weak Simulations). *A relation $\mathcal{R}$ on subdistributions over S is called a* (semi-)weak simulation *iff whenever $\mu_1 \mathcal{R} \mu_2$ then for all $\alpha \in \mathit{Act}^{\chi}$: $|\mu_1| = |\mu_2|$ and*

$\forall E \in \mathit{Supp}(\mu_1)$: $\exists {\mu_2}^g, {\mu_2}^s$: $({\mu_2}^g, {\mu_2}^s) \in \mathit{split}^{(\circ)}(\mu_2)$ and

(i) $[\![(E,\mu_1(E))]\!] \;\mathcal{R}\; {\mu_2}^g$ and $(\mu_1{-}E) \;\mathcal{R}\; {\mu_2}^s$

(ii) whenever $E \xrightarrow{\alpha} \mu_1'$ for some μ_1' then ${\mu_2}^g \xRightarrow{\hat{\alpha}}_C \mu''$ and $(\mu_1(E) \cdot \mu_1') \;\mathcal{R}\; \mu''$

Two subdistributions μ and γ are (semi-)weak similar, denoted by $\mu \precapprox^{(\circ)} \gamma$, if the pair (μ, γ) is contained in some (semi-)weak simulation.

It is not obvious that these relations are indeed equivalence relations and preorders, respectively. Reflexivity and symmetry is straightforward. The latter holds, because the union of two (semi-)weak bisimulations is again a (semi-)weak bisimulation. Since the pioneering work of Larsen and Skou [35], it has become a standard for bisimilarity notions defined in the stochastic setting, to presuppose the bisimulations to be equivalence relations on states. Because this property is not closed under union, several otherwise standard properties become difficult to establish. Owed to the distribution perspective on these relations illustrated above, the equivalence relation presupposition can be dropped, simplifying an easy exercise. Only transitivity needs a technical and involved proof. The proof for $\approx$ can be found in [22,23]. The crucial idea for this proof is that we can define (bi)simulation relations up-to-splitting. We refer the reader to [22] for further details. The proof for $\approx^\circ$ follows exactly the lines of that proof, but needs to distinguish fewer cases.

Lemma 2. *$\precapprox$ and $\precapprox^\circ$ are preorders, $\approx$ and $\approx^\circ$ are equivalence relations.*

It is apparent that $\approx$ and $\precapprox$ are weaker notions than $\approx^\circ$ and $\precapprox^\circ$ respectively:

Theorem 1. $\approx^\circ \subseteq \approx$ *and* $\precapprox^\circ \subseteq \precapprox$

The relations defined above relate subdistributions, but they induce relations on states in the obvious way: We call two states E, F (semi-)weak bisimilar, denoted by $E \approx_\Delta^{(\circ)} F$, if $\Delta_E \approx^{(\circ)} \Delta_F$. Analogously, we call two states E, F (semi-)weak similar, denoted by $E \precapprox_\Delta^{(\circ)} F$, if $\Delta_E \precapprox_\Delta^{(\circ)} \Delta_F$.

In the following we establish that $\asymp$ and $\approx_\Delta^\circ$ coincide. Since $\asymp$ is the naïve (state-based) integration of PA and IMC weak bisimulation, this fact provides insight into the twist achieved by moving from semi-weak to weak formulation.

Theorem 2. $\asymp \;=\; \approx^\circ_\Delta$

Proof. We first prove that the lifting of semi-weak bisimilarity to states, $\approx^\circ_\Delta$, is also a state-based bisimulation in the sense of Definition 10. The crucial point in this proof is the claim, that $\mu \approx^\circ \gamma$ implies $\forall C \in S/{\approx^\circ_\Delta} : \mu(C) = \gamma(C)$, since then the conditions of Definition 10 follow immediately. To see the claim, note that $\approx^\circ$ itself is a semi-weak bisimulation. Then, by repeated application of the left hand side of clause (i) in the definition of $\approx^\circ$, we can split γ into a family of subdistribution $\{\gamma_E\}_{E \in \mathit{Supp}(\mu)}$, such that every $E \in \mathit{Supp}(\mu)$ is matched by one of these distributions, and $\mu(E)\Delta_E \approx^\circ \gamma_E$ holds. In turn, we can split $\mu(E)\Delta_E$ into a family $\{\mu^E_F\}_{F \in \mathit{Supp}(\gamma_E)}$ accordingly, such that each state $F \in \mathit{Supp}(\gamma_E)$ is matched by the subdistribution μ^E_F , satisfying $\mu^E_F \approx^\circ \gamma_E(F)\Delta_F$. Every subdistributions μ^E_F must be of the form $\gamma_E(F)\Delta_E$. Hence, we know that $\gamma_E(F)\Delta_E \approx^\circ \gamma_E(F)\Delta_F$. In total, we have split μ and γ into sets of subdistributions, such that there is a total matching of subdistribution of one set with subdistributions of the other set. Matched subdistributions have the same size and the single elements of their supports are equivalent up to $\approx^\circ_\Delta$. From here we can immediately conclude that $\forall C \in S/{\approx^\circ_\Delta} : \mu(C) = \gamma(C)$ holds.

For the other direction we show that the relation

$$\mathcal{R} = \{(\mu, \gamma) \mid \forall C \in S/{\asymp} : \mu(C) = \gamma(C)\}$$

is a semi-weak bisimulation. Then, whenever $E \asymp F$, the pair (Δ_E, Δ_F) is contained in the semi-weak bisimulation $\mathcal{R}$, which implies $E \approx^\circ_\Delta F$. Let us consider an arbitrary pair $(\mu, \gamma) \in \mathcal{R}$. By symmetry it suffices to check the necessary conditions for an arbitrary $E \in \mathit{Supp}(\mu)$. Let C be the equivalence class of $\asymp$ containing E. Since $\mu(C) = \gamma(C)$, there exists a splitting $\gamma^g \oplus \gamma^s$ of γ with $\mathit{Supp}(\gamma^g) = \{F_1, \ldots, F_k\}$ and $F_i \asymp E$ for each F_i, and furthermore, $\forall C \in S/{\asymp} : \gamma^s(C) = (\mu - E)(C)$. Hence Condition (i) is satisfied. Whenever $E \xrightarrow{\alpha} \mu'$, following Condition (ii), then for each F_i we immediately deduce from $E \asymp F_i$ that $F_i \overset{\alpha}{\Longrightarrow}_C \gamma_{F_i}$ and $\forall C \in S/{\asymp} : \mu'(C) = \gamma_{F_i}(C)$. Let us set $\rho := \bigoplus_{i=1\ldots k} \gamma_{F_i}$. It is then straightforward to show that in total $\gamma^g \overset{\alpha}{\Longrightarrow}_C \rho$ and that $\forall C \in S/{\asymp} : \mu(E) \cdot \mu'(C) = \rho(C)$. By the choice of $\mathcal{R}$ this immediately implies $(\mu(E)\mu', \rho) \in \mathcal{R}$, which suffices to establish Condition (ii). □

Just like $\asymp$, most existing weak relations for systems with probabilistic or stochastic timed transitions can be recast as relations on distributions, which can be formulated as slight adaptations of $\approx^\circ$ and $\precsim^\circ$, respectively. So, with $\approx^\circ$ and $\precsim^\circ$ at hand, the exact extra power of the distribution-based perspective, combined with distribution splitting, becomes apparent: $\approx^\circ$ and $\approx$ only differ in their use of *split*$^\circ(\mu)$ and *split*(μ), respectively. The latter allows additional internal transition sequences, and is the key to fuse distributions along sequences thereof. It is thus a natural generalisation comparable to the classical passage from strong transitions to weak transitions.

Discussion. We will now summarise the relationship between the respective standard notions of weak (bi-)similarity on the submodels and weak (bi-)similarity for Markov automata. Since all of these relations are defined over states, we will compare them to $\approx_\Delta$ and $\approx^\circ_\Delta$.

1. On *labelled transitions systems*, both $\approx_\Delta$ and $\approx^\circ_\Delta$ coincide with stable weak bisimilarity [49]. They both coincide with standard weak bisimilarity [40] if no infinite sequences of internal transitions appear in the LTS. This difference is inherited from IMC, and owed to the maximal progress assumption [28]. The same applies to $\precapprox_\Delta$, $\precapprox^\circ_\Delta$, and weak similarity on LTS [40].
2. On *discrete-time Markov chains*, $\approx_\Delta$ and $\approx^\circ_\Delta$ coincide with weak bisimilarity [3,5]. We claim that $\precapprox_\Delta$ and $\precapprox^\circ_\Delta$ can be adapted such that they both coincide with weak similarity for state-labelled DTMC [5].
3. On *continuous-time Markov chains*, $\approx_\Delta$ and $\approx^\circ_\Delta$ coincide with lumping equivalence [26], due to the fact that our weak transitions do not affect timed transitions. For this reason we see no obvious way to adapt $\precapprox_\Delta$ and $\precapprox^\circ_\Delta$ such that they match weak similarity for state-labelled CTMC [5].
4. On *probabilistic automata*, $\approx^\circ_\Delta$ coincides with weak bisimilarity [44], if restricting to models without infinite sequences of internal transition. This slight restriction is again a legacy of the maximal progress assumption. This technical discrepancy carries over to all other relations defined on PA. If instead we adapt the definition and remove the stability condition, the adapted version of $\approx^\circ_\Delta$ and weak bisimilarity on PA [44] coincide. The same holds for $\precapprox^\circ_\Delta$ and weak similarity for PA [44]. Remarkably, $\precapprox_\Delta$ and probabilistic forward similarity $\leq_{fwd}$ [44] coincide.
5. On *interactive Markov chains* $\approx_\Delta$ and $\approx^\circ_\Delta$ coincide with weak bisimulation [25]. A weaker variant is found in [10]. To the best of our knowledge no weak similarity relations for IMC have been introduced in the literature so far, so the one jointly induced by $\precapprox$ and $\precapprox^\circ$ is new.

The fact that on PA $\precapprox_\Delta$ and $\leq_{fwd}$ agree is especially interesting, since a bisimulation variant of this relation was not known to date, but is now at hand with $\approx_\Delta$. Furthermore, $\approx_\Delta$ has a selection of distinguishing properties. We refer the reader to [22] for details. We mention only briefly, that $\approx_\Delta$ is a congruence with respect to parallel composition. The congruence property can be established for other standard process algebraic operators – with the usual root condition being needed to arrive at a congruence for non-deterministic choice.

We finally want to correct our claim [22], that a reformulation of PA weak bisimilarity as a relation on distribution would not be compositional with respect to subdistribution composition, now turns out to be wrong. It is easy to show that $\approx^\circ$ is indeed compositional with respect to this operator, and since on PA, $\approx^\circ$ coincides with PA weak bisimilarity (except for divergence behaviour), this also holds for PA weak bisimilarity.

6 Conclusions

This paper has tried to provide insight into the foundational aspects of Markov automata, a model that integrates probabilistic automata and interactive Markov chains. We have laid out the principal ingredients of a compositional theory for *MA*, and have discussed how a lifting of relations to (sub)distributions, together with the notion of distribution splitting, enables us to cast a variety of existing simulations and bisimulations in a uniform setting, making subtle differences and semantic choices apparent.

Markov automata target a domain of concurrency modelling and evaluation, where designers find it adequate to work with durations that are memoryless, and need to represent branching probabilities as well as concurrency in a convenient manner. In this area, GSPNs have seen broad applicability, but, as we have highlighted, only with incomplete semantic understanding. The *MA* model changes the picture, it can serve as a semantic foundation for GSPN, and, since it is compositional in a strict sense, we think it is the nucleus for a fully compositional and usable language for this modelling domain. Noteworthy, the *MA* model is – just like IMC – rich enough to allow for non-exponential distributions, namely by approximating them as phase-type distributions [27], albeit at the price of a state space increase.

PA as well as IMC are supported by mature software tools, PRISM [29] and CADP [16]. We are eagerly exploring possibilities to arrive at tool support for the analysis of *MA*.

In this paper, we have restricted our attention to finite and finitely-branching models. It remains for further work to establish the results of this paper in a setting with (un)countably many states or transitions.

Acknowledgements. Christian Eisentraut and Holger Hermanns are supported by the DFG as part of the Transregional Collaborative Research Centre SFB/TR 14 *AVACS*, by the European Community's Seventh Framework Programme under grant agreement n^o 214755 *QUASIMODO*, and by the NWO-DFG bilateral project *ROCKS*. The work of Lijun Zhang has received partial support from *MT-LAB*, a VKR Centre of Excellence.

References

1. Abadi, M., Lamport, L.: The existence of refinement mappings. Theoretical Computer Science 82(2), 253–284 (1991)
2. Baier, C., Engelen, B., Majster-Cederbaum, M.E.: Deciding bisimilarity and similarity for probabilistic processes. J. Comput. Syst. Sci. 60(1), 187–231 (2000)
3. Baier, C., Hermanns, H.: Weak bisimulation for fully probabilistic processes. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 119–130. Springer, Heidelberg (1997)
4. Baier, C., Katoen, J.-P.: Principles of Model Checking, 1st edn. MIT Press, Cambridge (2008)
5. Baier, C., Katoen, J.-P., Hermanns, H., Wolf, V.: Comparative branching-time semantics for Markov chains. Information and Computation 200(2), 149–214 (2005)
6. Balbo, G.: Introduction to generalized stochastic Petri nets. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 83–131. Springer, Heidelberg (2007)
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. Bergstra, J.A., Ponse, A., Smolka, S.A.: Handbook of Process Algebra. Elsevier, Amsterdam (2001)
9. Bolognesi, T., Brinksma, E.: Introduction to the iso specification language lotos. Computer Networks 14, 25–59 (1987)
10. Bravetti, M.: Revisiting interactive markov chains. Electr. Notes Theor. Comput. Sci. 68(5) (2002)
11. Busch, H., Sandmann, W., Wolf, V.: A numerical aggregation algorithm for the enzyme-catalyzed substrate conversion. In: Priami, C. (ed.) CMSB 2006. LNCS (LNBI), vol. 4210, pp. 298–311. Springer, Heidelberg (2006)

12. Cheung, L., Stoelinga, M., Vaandrager, F.W.: A testing scenario for probabilistic processes. J. ACM 54(6) (2007)
13. Chiola, G., Donatelli, S., Franceschinis, G.: GSPNs versus SPNs: What is the actual role of immediate transitions? In: PNPM, pp. 20–31. IEEE, Los Alamitos (1991)
14. Chiola, G., Franceschinis, G., Gaeta, R., Ribaudo, M.: GreatSPN 1.7: Graphical editor and analyzer for timed and stochastic Petri nets. Perf. Eval. 24(1-2), 47–68 (1995)
15. 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(2), 89–107 (1993)
16. Coste, N., Garavel, H., Hermanns, H., Hersemeule, R., Thonnart, Y., Zidouni, M.: Quantitative evaluation in embedded system design: Validation of multiprocessor multithreaded architectures. In: DATE, pp. 88–89 (2008)
17. Courtney, T., Daly, D., Derisavi, S., Gaonkar, S., Griffith, M., Lam, V.V., Sanders, W.H.: The Möbius modeling environment: Recent developments. In: QEST, pp. 328–329. IEEE, Los Alamitos (2004)
18. Deng, Y., van Glabbeek, R.J., Hennessy, M., Morgan, C.: Testing finitary probabilistic processes. In: CONCUR, pp. 274–288 (2009)
19. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Approximating labelled Markov processes. Inf. Comput. 184(1), 160–200 (2003)
20. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Weak bisimulation is sound and complete for PCTL*. Inf. Comput. 208(2), 203–219 (2010)
21. Desharnais, J., Laviolette, F., Tracol, M.: Approximate analysis of probabilistic processes: Logic, simulation and games. In: QEST, pp. 264–273 (2008)
22. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuos time. In: LICS. IEEE, Los Alamitos (to appear, 2010)
23. Eisentraut, C., Hermanns, H., Zhang, L.: Probabilistic automata in continuous time. Reports of SFB/TR 14 AVACS 62, Saarland University (April 2010), http://www.avacs.org
24. Florio, V.D., Donatelli, S., Dondossola, G.: Flexible development of dependability services: An experience derived from energy automation systems. In: ECBS, pp. 86–93. IEEE, Los Alamitos (2002)
25. Hermanns, H. (ed.): Interactive Markov Chains. LNCS, vol. 2428, p. 57. Springer, Heidelberg (2002)
26. Hermanns, H., Herzog, U., Katoen, J.-P.: Process algebra for performance evaluation. Theoretical Computer Science 274(1-2), 43–87 (2002)
27. Hermanns, H., Katoen, J.-P.: Automated compositional Markov chain generation for a plain-old telephone system. Science of Comp. Progr. 36(1), 97–127 (2000)
28. Hermanns, H., Lohrey, M.: Priority and maximal progress are completely axioatisable (extended abstract). In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 237–252. Springer, Heidelberg (1998)
29. Hinton, A., Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
30. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, Englewood Cliffs (1985)
31. Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: LICS, pp. 266–277. IEEE, Los Alamitos (1991)
32. Katoen, J.-P., Klink, D., Neuhäußer, M.R.: Compositional abstraction for stochastic systems. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 195–211. Springer, Heidelberg (2009)
33. Kemeny, J.G., Snell, J.L., Knapp, A.W.: Denumerable Markov Chains, 2nd edn. Springer, Heidelberg (1976)

34. Klin, B., Sassone, V.: Structural operational semantics for stochastic process calculi. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 428–442. Springer, Heidelberg (2008)
35. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Information and Computation 94(1), 1–28 (1991)
36. Lynch, N.A., Tuttle, M.R.: Hierarchical correctness proofs for distributed algorithms. In: PODC, pp. 137–151 (1987)
37. Marsan, M.A., Balbo, G., Chiola, G., Conte, G.: Generalized stochastic Petri nets revisited: Random switches and priorities. In: PNPM, pp. 44–53. IEEE, Los Alamitos (1987)
38. Marsan, M.A., Balbo, G., Chiola, G., Conte, G., Donatelli, S., Franceschinis, G.: An introduction to generalized stochastic Petri nets. Microel. and Rel. 31(4), 699–725 (1991)
39. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley & Sons, Chichester (1995)
40. Milner, R.: Communication and Concurrency. Prentice Hall, Englewood Cliffs (1989)
41. Palamidessi, C.: Comparing the expressive power of the synchronous and asynchronous pi-calculi. Mathematical Structures in Computer Science 13(5), 685–719 (2003)
42. Park, D.M.R.: Concurrency and automata on infinite sequences. Theoretical Computer Science, pp. 167–183 (1981)
43. Sandmann, W., Wolf, V.: Computational probability for systems biology. In: Fisher, J. (ed.) FMSB 2008. LNCS (LNBI), vol. 5054, pp. 33–47. Springer, Heidelberg (2008)
44. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology (1995)
45. Segala, R.: Probability and nondeterminism in operational models of concurrency. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 64–78. Springer, Heidelberg (2006)
46. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing 2(2), 250–273 (1995)
47. Stewart, W.J.: Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling. Princeton University Press, Princeton (2009)
48. Stoelinga, M., Vaandrager, F.W.: Root contention in IEEE 1394. In: Katoen, J.-P. (ed.) AMAST-ARTS 1999, ARTS 1999, and AMAST-WS 1999. LNCS, vol. 1601, pp. 53–74. Springer, Heidelberg (1999)
49. van Glabbeek, R.J.: The linear time - branching time spectrum II. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 66–81. Springer, Heidelberg (1993)
50. Wolf, V.: Modelling of biochemical reactions by stochastic automata networks. ENTCS 171(2), 197–208 (2007)
51. Zhang, L.: Decision Algorithms for Probabilistic Simulations. PhD thesis, Universität des Saarlandes (2008)
52. Zhang, L., Hermanns, H., Eisenbrand, F., Jansen, D.N.: Flow faster: Efficient decision algorithms for probabilistic simulations. Logical Methods in Computer Science 4(4) (2008)

Taming Distributed Asynchronous Systems

Anca Muscholl

LaBRI, University Bordeaux, France

Abstract. This extended abstract surveys some analysis techniques for distributed, asynchronous systems with two kinds of synchronization, shared variables and fifo channels.

1 Introduction

Modeling distributed, asynchronous systems so that computer-assisted analysis becomes feasible, is an on-going challenge in both theory and practice. Several automata-based models for such systems have been proposed and studied over the past twenty years, capturing various aspects of distributed behavior. Depending on the motivation, such models fall into two large categories. In the first one we find rather simple models, capturing basic synchronization mechanisms: Petri nets, communicating automata, ... They were studied for algorithmic properties and/or their expressive power. In the second category we see more sophisticated models, that were conceived for supporting practical system design, like Harel's statecharts, or Lynch's I/O automata. It is clear that being able to develop automated verification techniques requires a good understanding of the simpler models, in particular since more complex ones are often built as a combination of basic models.

In this survey we address the issue of analyzing networks of (mostly finite-state) processes with two kinds of communication mechanisms, unbounded fifo channels and shared variables. We also go one step beyond verification, or model-checking, by addressing the synthesis problem in the shared-variable case. Synthesis, and in particular controller synthesis, is a challenging problem even for such simple models as the ones considered in this survey, since it essentially amounts to solve distributed games. This topic is still rather poorly understood and open for future research, in spite of considerable efforts and partial results obtained during the past decade.

2 Models of Distributed Computation

The architecture of a distributed asynchronous system consists of a set of processes $\mathcal{P}$ related by links, and we will consider it as *fixed*. Such links may correspond for instance to communication channels or to shared variables. We do not discuss here other synchronization mechanisms that appear in the literature, like e.g. state observation or signals.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 40–47, 2010.

Zielonka's *asynchronous automata* is an asynchronous model based on shared variables. It has its roots in the theory of Mazurkiewicz traces [28], which came up in the late seventies in connection with the semantics of 1-safe Petri nets (the reader may find in [11] a wealth of results about traces). Asynchronous automata provide one of the first highly non-trivial examples of distributed (closed) synthesis, as expressed in Theorem 1 below.

Given a finite set $\mathcal{P}$ of processes, we consider an alphabet of actions Σ and a location function $dom : \Sigma \to (2^{\mathcal{P}} \setminus \emptyset)$, associating with each action a non-empty set of processes. The location mapping *dom* defines in a natural way an *independence relation* I: two actions $a, b \in \Sigma$ are independent (denoted as $(a, b) \in I$) if they synchronize disjoint sets of processes, i.e., if $dom(a) \cap dom(b) = \emptyset$. One can define the relation $\sim_I$ on Σ^* as the equivalence generated by all pairs $(uabv, ubav)$, for $(a, b) \in I$ and $u, v \in \Sigma^*$. A *trace* is then a $\sim_I$-equivalence class, and a trace language is a word language closed under $\sim_I$.

Alternatively, traces can be viewed as labeled pomsets (see an example in Figure 1), and the set of (labeled) linearizations of such a pomset corresponds to the $\sim_I$-equivalence class $[u]$ of any of these linearizations $u \in \Sigma^*$.

A (deterministic) *asynchronous automaton* is a tuple

$$\mathcal{A} = \langle \{S_p\}_{p \in \mathcal{P}}, s_0, \{\delta_a\}_{a \in \Sigma}, F \rangle ,$$

where

- S_p is a finite set of (local) states of process p,
- $s_0 \in \prod_{p \in \mathbb{P}} S_p$ is a (global) initial state,
- $\delta_a : \prod_{p \in dom(a)} S_p \to \prod_{p \in dom(a)} S_p$ is a transition relation; so on a letter $a \in \Sigma$ it is a partial function on tuples of states of processes in $dom(a)$,
- $F \subseteq \prod_{p \in \mathcal{P}} S_p$ is a set of final (accepting) states.

An asynchronous automaton can be seen as a sequential automaton with the state set $S = \prod_{p \in \mathcal{P}} S_p$ and transitions $s \xrightarrow{a} s'$ if $((s_p)_{p \in dom(a)}, (s'_p)_{p \in dom(a)}) \in \delta_a$, and $s_q = s'_q$ for all $q \notin dom(a)$. By $L(\mathcal{A})$ we denote the set of words labeling accepting runs. This definition has an important consequence. If $(a, b) \in I$ then the same state is reached on the words ab and ba. More generally, whenever $u \sim_I v$ and $u \in L(\mathcal{A})$ then $v \in L(\mathcal{A})$, too. This means that $L(\mathcal{A})$ is a trace language.

Example 1. Let us consider the asynchronous automaton $\mathcal{A}$ defined by $S_p = \{0\}$, $S_q = S_r = \{0, 1\}$, and transition function $\delta_a(s_p, s_q) = (s_p, \neg s_q)$ if $s_q = 1$ (undefined otherwise), $\delta_d(s_r) = \neg s_r$ if $s_r = 1$ (undefined otherwise), $\delta_b(s_q, s_r) = (1, 1)$ if $s_q \wedge s_r = 0$ (undefined otherwise) and $\delta_c(s_p) = s_p$. Starting with $s_0 = (0, 0, 0)$, an accepting run of $\mathcal{A}$ checks that between any two successive b-events, there is either an a or a d (or both), and there is a b-event before all a and d.

One of the deepest results of trace theory is Zielonka's construction of a deterministic asynchronous automaton from a finite-state one. One can see it as an example of distributed *closed* synthesis, i.e., without any environment.

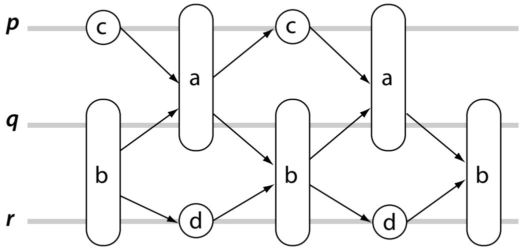


Fig. 1. The pomset associated with the trace $t = [c\,b\,a\,d\,c\,b\,a\,d\,b]$, with $dom(a) = \{p, q\}$, $dom(b) = \{q, r\}$, $dom(c) = \{p\}$, $dom(d) = \{r\}$.

Theorem 1. *[40] Given a finite automaton $\mathcal{A}$ accepting the trace language $L(\mathcal{A})$, a deterministic asynchronous automaton $\mathcal{B}$ can be effectively constructed with $L(\mathcal{A}) = L(\mathcal{B})$.*

The above construction has received a lot of interest, and a series of papers aimed at improving it algorithmically (see e.g. [10,30,19,17]). Currently the best construction starting with a DFA $\mathcal{A}$ is polynomial in the size of $\mathcal{A}$ and simply exponential in the number of processes. Surprisingly, it is rather difficult to come up with a matching lower bound (see [17] for partial results). As explained in Section 3, this construction plays a fundamental role in other settings of distributed synthesis, as for instance for communicating automata, that we present next.

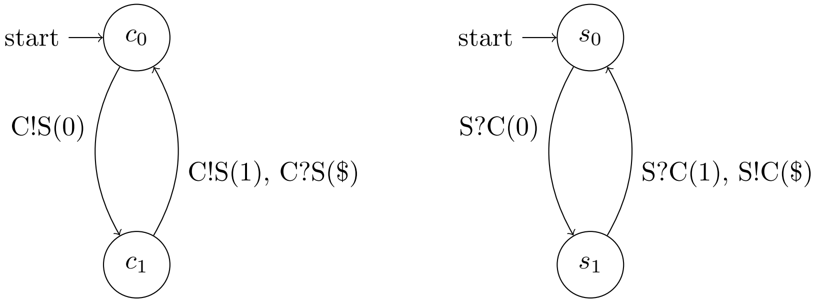
A communicating automaton (*CA* for short) is parametrized by a set $\mathcal{P}$ of processes, a set of point-to-point fifo channels $Ch \subseteq \mathcal{P}^2 \setminus id_{\mathcal{P}}$, and a set of message contents *Msg*. It is a tuple $\mathcal{A} = \langle (\mathcal{A}_p)_{p \in \mathcal{P}}, \Sigma, F \rangle$ where

– each $\mathcal{A}_p = (S_p, \rightarrow_p, s_p^0)$ is a finite labeled transition system with state space S_p, transition relation $\rightarrow_p \subseteq S_p \times \Sigma_p \times S_p$, and initial state $s_p^0 \in S_p$; the local action alphabet Σ_p consists of send actions (denoted as $p!q(m)$, with $(p, q) \in Ch$, $m \in Msg$), receive actions (denoted as $p?r(m)$, with $(r, p) \in Ch$, $m \in Msg$), and local actions.
– $F \subseteq \prod_{p \in \mathcal{P}} S_p$ is a set of *global* final states.

We denote the product $S := \prod_{p \in \mathcal{P}} S_p$ as set of *global states.*

The behavior of a CA is defined as the behavior of an infinite labeled transition system, by considering the possible (local) transitions on the set of configurations of the CA. A *configuration* of the CA $\mathcal{A}$ consists of a global state, together with a word from Msg^* for each channel $(p, q) \in Ch$. Transitions are defined in the usual way: the effect of an action $a \in \Sigma_p$ is to change the S_p state component according to $\mathcal{A}_p$, and to perform the obvious modification on one channel of p, according to a being a send of message m from p to q (written as $a = p!q(m)$) or a receive of m on p from r (written as $a = p?r(m)$).

Example 2. The CA in the figure below describes the communication between two (finite-state) processes C and S, connected through one channel in each direction. The set of message contents is $Msg = \{0, 1, \$\}$. From the initial configuration $\langle (c_0, s_0), (\varepsilon, \varepsilon)\rangle$ (say, (C, S) is the first channel) one can reach e.g. the configurations $\langle (c_1, s_0), (010, \varepsilon)\rangle$ and $\langle (c_0, s_0), (101, \$)\rangle$, but not $\langle (c_0, s_0), (0101, \$)\rangle$. For instance, $\langle (c_0, s_0), (\varepsilon, \varepsilon)\rangle \xrightarrow{C!S(0)} \langle (c_1, s_0), (0, \varepsilon)\rangle \xrightarrow{C!S(1)} \langle (c_0, s_0), (01, \varepsilon)\rangle \xrightarrow{C!S(0)} \langle (c_1, s_0), (010, \varepsilon)\rangle$.



Like traces being partially ordered representations of runs of asynchronous automata, runs of CA have a natural interpretation in terms of labeled pomsets, too. The pomsets associated with such runs are called *message sequence charts*, and represent in a diagrammatic way messages exchanged between processes.

3 Analyzing Communicating Automata

In spite of their simplicity, communicating automata are Turing-powerful, as it can be easily seen (by simulating e.g. Post tag systems). From the verification viewpoint this immediately implies that one needs to accept approximated or semi-algorithmic solutions.

Simple approximated solutions, like ignoring the order of messages in the channels or imposing a limit on their size, are of course too coarse. Acceleration methods using some finitary representation of possibly infinite sets of configurations (called symbolic representations), are a more powerful example of under-approximation. In the case of communicating automata, such symbolic representations are based on finite automata or some extended automata models with good algorithmic properties [4,5,7]. The general idea is to speed-up the naive enumeration of reachable configurations, by computing the result of loop iteration.

A nice example for over-approximating methods are lossy channel systems. Of course, such a model may be interesting in its own right, since it allows to model imperfect channels. Lossy channels are a particular instance of well-structured transition systems [13,2]. In particular, questions like control-state reachability and termination are decidable [2,14], albeit of non-primitive recursive complexity [36]. On the other hand, liveness properties or boundedness of lossy channels are undecidable [1,27].

Whereas the above mentioned approaches emphasize symbolic representations of sets of (reachable) configurations, there is a complementary, language-oriented approach based on *partial orders*. The language-theoretical viewpoint emphasizes the (partially-ordered) executions, instead of the channel contents. This kind of event-based reasoning arises very naturally when communicating automata are viewed as sequential automata synchronizing over communication events. The main advantage it offers is that the synthesis problem can be stated in a natural way.

Undecidability of various questions about communicating automata has actually two sources: the first, obvious one, is the unboundedness of channels. The second, more subtle, comes up when the specification formalism (e.g. regular ones like LTL) is incompatible with the partially-ordered model. As a consequence, getting solutions for model-checking or synthesis requires both *channel restrictions* and *partial order specifications*.

A universally channel-bounded automaton is one where there is a uniform bound on the size of channels, over all reachable configurations. So a universally bounded automaton is just a finite state system. A much less restrictive notion is an existential channel-bound. Such a bound roughly means that any execution can be *rescheduled* in such a way that it can be executed with bounded channels. In particular, existential bounds admit channels of arbitrary size. A simple example illustrating the idea is a pair of processes, a producer and a consumer, where the producer keeps sending messages to the consumer, who is supposed to accept every message. Since there is no control on the relative speed of these two processes, there is no bound on the number of messages in transit. But for verifying many properties, like e.g. control-state reachability, it suffices to reason about schedulings where messages are consumed without delay, i.e. where executions can be scheduled with a channel of size one.

The main result obtained in this setting is a solution for closed synthesis, that can be stated as a Kleene-Büchi theorem about communicating automata with channel bounds [21,18]. A main ingredient of these constructions is the link between automata with channel bounds and trace languages and in particular, Zielonka's construction of asynchronous (trace) automata. Model-checking existentially bounded automata w.r.t. partial order specifications like MSO [24], closed regular specifications [20] or PDL [6], is also decidable.

Several promising, recent research directions can be mentioned. One of them is motivated by the need of analyzing distributed recursive programs, and aims at identifying reasonable, tractable subclasses of communicating automata extended by additional capabilities for the single processes, like for instance pushdown storage [3,39,22]. A second, quite challenging perspective for future work is the general synthesis problem for communicating systems. This problem can be stated in many different ways, depending on the degree of completeness of the specification (specifications may e.g. talk only about external messages). However, one probably needs first a solution for the problem described in the next section, before working out a general solution for synthesizing or controlling communicating automata.

4 Distributed Control for Asynchronous Automata

In the simplest case, the synthesis problem asks to find a model for a given specification, so it is just a satisfiability problem, where one is given some formalism for the specification (e.g. logics) and one for the model (e.g. finite automata). In a more refined version one is also given a system (or plant, usually an automaton) and is asked to find a controller such that the controlled system satisfies the given specification. The control consists in forbidding some actions of the plant, but not every action can be forbidden, and the control has also to ensure that the system does not block completely.

Synthesis was first considered in a synchronous (hardware) setting by Church [9]. In his model, the specification is a relation between input variables, which are controlled by the environment, and output variables, controlled by the (centralized) system. Church's problem stimulated a fruitful research direction on 2-person, zero-sum infinitary games, starting with the fundamental results of [8, 34,35] (see also [37,38] for recent surveys).

Distributed controller synthesis is a more recent research topic, that was initiated by Pnueli and Rosner [33], who show that only very restricted architectures admit a decidable synthesis problem. Undecidability of distributed synthesis follows already from the work of Peterson and Reif on "multiple-person alternating machines" [32].

Various versions of distributed synthesis appear in the literature. One important distinction is to be made between synchronous and asynchronous systems, respectively. In the synchronous case, processes execute a step at each (global) clock tick, whereas in the asynchronous case they are decoupled. Another distinction is how much information is allowed to be exchanged between processes. At least two different classes of models were studied here. In the model considered by [33,23,12,16], a distributed system is given by an architecture describing (synchronous) channels between processes, and the information conveyed via the channels between processes, is finite. In the model studied in [15,26,31], the distributed system is an asynchronous automaton (and the controller is also required to be such an automaton). Here, the information exchanged between processes corresponds to the causal past of events, therefore it is unbounded.

Decidability for synchronous synthesis basically requires a pipeline architecture where information flows in a single direction (see [33,23,25,29,12,16] for various refinements). To state it informally, the reasons for undecidability are either global specifications or "information forks", like the case where two independent processes can be "observed" by a third one.

Compared with the synchronous case, our understanding of asynchronous controller synthesis is still unsatisfactory. For instance, it is open whether this problem is decidable! Two decidability results are known in this setting. The first one [15] was obtained by restricting the (in)dependencies between letters of the input alphabet. The second paper [26] shows decidability by restricting the plant: roughly speaking, the restriction requires that if two processes do not synchronize during a long amount of time, then they won't synchronize ever again. The proof of [26] goes beyond the controller synthesis problem, by coding it into monadic second-order theory of event structures and showing that this theory is decidable when the criterion on the asynchronous automaton holds.

References

1. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.-K.: General decidability theorems for infinite state systems. In: LICS 1996, pp. 313–323. IEEE Computer Society, Los Alamitos (1996)
2. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Inform. and Comput. 127(2), 91–101 (1996)
3. Atig, M.F., Bouajjani, A., Touili, T.: On the reachability analysis of acyclic networks of pushdown systems. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 356–371. Springer, Heidelberg (2008)
4. Boigelot, B., Godefroid, P.: Symbolic verification of communication protocols with infinite state spaces using QDDs. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 1–12. Springer, Heidelberg (1996)
5. Boigelot, B., Godefroid, P., Willems, B., Wolper, P.: The power of QDDs. In: Van Hentenryck, P. (ed.) SAS 1997. LNCS, vol. 1302, pp. 172–186. Springer, Heidelberg (1997)
6. Bollig, B., Kuske, D., Meinecke, I.: Propositional Dynamic Logic for message-passing systems. Logical Methods in Computer Science (to appear, 2010)
7. Bouajjani, A., Habermehl, P.: Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. Theor. Comp. Science 221(1-2), 211–250 (1999)
8. Büchi, J., Landweber, L.: Definability in the monadic second order theory of successor. J. of Symb. Logic 34(2), 166–170 (1969)
9. Church, A.: Applications of recursive arithmetic to the problem of circuit synthesis. In: Summaries of the Summer Institute of Symbolic Logic, vol. 1, pp. 3–50. Cornell Univ. (1957)
10. Cori, R., Métivier, Y., Zielonka, W.: Asynchronous mappings and asynchronous cellular automata. Inform. and Comput. 106, 159–202 (1993)
11. Diekert, V., Rozenberg, G. (eds.): The Book of Traces. World Scientific, Singapore (1995)
12. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: LICS 2005, pp. 321–330. IEEE Computer Society Press, Los Alamitos (2005)
13. Finkel, A.: A generalization of the procedure of Karp and Miller to well structured transition systems. In: Ottmann, T. (ed.) ICALP 1987. LNCS, vol. 267, pp. 499–508. Springer, Heidelberg (1987)
14. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theor. Comp. Science 256(1-2), 63–92 (2001)
15. Gastin, P., Lerman, B., Zeitoun, M.: Distributed games with causal memory are decidable for series-parallel systems. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 275–286. Springer, Heidelberg (2004)
16. Gastin, P., Sznajder, N., Zeitoun, M.: Distributed synthesis for well-connected architectures. Formal Methods in System Design 34(3), 215–237 (2009)
17. Genest, B., Gimbert, H., Muscholl, A., Walukiewicz, I.: Optimal Zielonka-type construction of deterministic asynchronous automata. In: ICALP 2010. LNCS. Springer, Heidelberg (2010)
18. Genest, B., Kuske, D., Muscholl, A.: A Kleene theorem and model checking algorithms for existentially bounded communicating automata. Inform. and Comput. 204(6), 920–956 (2006)
19. Genest, B., Muscholl, A.: Constructing exponential-size deterministic Zielonka automata. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 565–576. Springer, Heidelberg (2006)

20. Genest, B., Muscholl, A., Seidl, H., Zeitoun, M.: Infinite-state high-level MSCs: Model-checking and realizability. J. Comput. Syst. Sci. 72(4), 617–647 (2006)
21. Henriksen, J.G., Mukund, M., Kumar, K.N., Sohoni, M., Thiagarajan, P.: A theory of regular MSC languages. Inform. and Comput. 202(1), 1–38 (2005)
22. Heußner, A., Leroux, J., Muscholl, A., Sutre, G.: Reachability analysis of communicating pushdown systems. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 267–281. Springer, Heidelberg (2010)
23. Kupferman, O., Vardi, M.: Synthesizing distributed systems. In: LICS 2001. IEEE Computer Society Press, Los Alamitos (2001)
24. Madhusudan, P., Meenakshi, B.: Beyond message sequence graphs. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2001. LNCS, vol. 2245, pp. 256–267. Springer, Heidelberg (2001)
25. Madhusudan, P., Thiagarajan, P.: Distributed control and synthesis for local specifications. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 396–407. Springer, Heidelberg (2001)
26. Madhusudan, P., Thiagarajan, P.S., Yang, S.: The MSO theory of connectedly communicating processes. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 201–212. Springer, Heidelberg (2005)
27. Mayr, R.: Undecidable problems in unreliable computations. Theor. Comp. Science 297(1-3), 337–354 (2003)
28. Mazurkiewicz, A.: Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus (1977)
29. Mohalik, S., Walukiewicz, I.: Distributed games. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 338–351. Springer, Heidelberg (2003)
30. Mukund, M., Sohoni, M.: Gossiping, asynchronous automata and Zielonka's theorem. Report TCS-94-2, School of Mathematics, SPIC Science Foundation, Madras, India (1994)
31. Muscholl, A., Walukiewicz, I., Zeitoun, M.: A look at the control of asynchronous automata. In: Perspectives in Concurrency Theory. IARCS-Universities, Universities Press (2009)
32. Peterson, G.L., Reif, J.H.: Multi-person alternation. In: FOCS 1979, pp. 348–363. IEEE Computer Society Press, Los Alamitos (1979)
33. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: FOCS 1990, pp. 746–757. IEEE Computer Society Press, Los Alamitos (1990)
34. Rabin, M.: Decidability of second-order theories and automata on infinite trees. Trans. Amer. Math. Soc. 141, 1–35 (1969)
35. Rabin, M.O.: Automata on Infinite Objects and Church's Problem. American Mathematical Society, Providence (1972)
36. Schnoebelen, P.: Verifying lossy channel systems has nonprimitive recursive complexity. Inform. Proc. Lett. 83(5), 251–261 (2002)
37. Thomas, W.: On the synthesis of strategies in infinite games. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 1–13. Springer, Heidelberg (1995)
38. Thomas, W.: Church's problem and a tour through automata theory. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.) Pillars of Computer Science. LNCS, vol. 4800, pp. 635–655. Springer, Heidelberg (2008)
39. Torre, S.L., Madhusudan, P., Parlato, G.: Context-bounded analysis of concurrent queue systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 299–314. Springer, Heidelberg (2008)
40. Zielonka, W.: Notes on finite asynchronous automata. RAIRO - Informatique Théorique et Applications 21, 99–135 (1987)

Trust in Anonymity Networks

Vladimiro Sassone, Sardaouna Hamadou, and Mu Yang

ECS, University of Southampton

Abstract. Anonymity is a security property of paramount importance, as we move steadily towards a wired, online community. Its import touches upon subjects as different as eGovernance, eBusiness and eLeisure, as well as personal freedom of speech in authoritarian societies. Trust metrics are used in anonymity networks to support and enhance reliability in the absence of verifiable identities, and a variety of security attacks currently focus on degrading a user's trustworthiness in the eyes of the other users. In this paper, we analyse the privacy guarantees of the Crowds anonymity protocol, with and without onion forwarding, for standard and adaptive attacks against the trust level of honest users.

1 Introduction

Protecting online privacy is an essential part of today's society and its importance is increasingly recognised as crucial in many fields of computer-aided human activity, such as eVoting, eAuctions, bill payments, online betting and electronic communication. One of the most common mechanisms for privacy is *anonymity*, which generally refers to the condition of being unidentifiable within a given set of subjects, known as the *anonymity set*.

Many schemes have been proposed to enforce privacy through anonymity networks (e.g. [6, 15, 19, 24, 25]). Yet, the open nature of such networks and the unaccountability which results from the very idea of anonymity, make the existing systems prone to various attacks (e.g. [10, 18, 22, 23]). An honest user may have to suffer repeated misbehaviour (e.g., receiving infected files) without being able to identify the malicious perpetrator. Keeping users anonymous also conceals their trustworthiness, which in turn makes the information exchanged through system transactions untrustworthy as well. Consequently, a considerable amount of research has recently been focussing on the development of trust-and-reputation-based metrics aimed at enhancing the reliability of anonymity networks [7–9, 11, 31, 33].

Developing an appropriate trust metric for anonymity is very challenging, due to the fact that trust and anonymity are seemingly conflicting notions. Consider for instance the trust networks of Figure 1. In (a) peer A trusts B and D, who both trust C. Assume now that C wants to request a service from A *anonymously*, by proving her trustworthiness to A (i.e., the existence of a trust link to it). If C can prove that she is trusted by D without revealing her identity (using e.g. a *zero-knowledge* proof [3]), then A cannot distinguish whether the request originated from C or E. Yet, A's trust in D could be insufficient to obtain that specific service from A. Therefore, C could strengthen her request by proving that she is trusted by both D and B. This increases the trust guarantee. Unfortunately, it also decreases C's anonymity, as A can compute the intersection of

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 48–70, 2010.

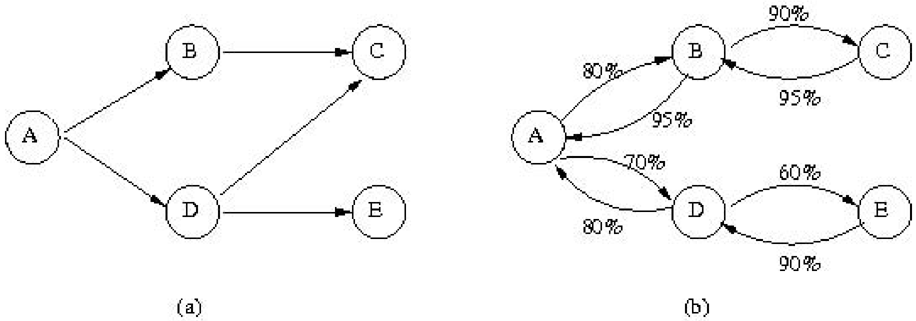


Fig. 1. Trust networks [3]

peers trusted by both D and B, and therefore restrict the range of possible identities for the request's originator, or even identify C uniquely. Indeed, consider Figure 1(b). Here the trust level between two principals is weighted, and trust between two non-adjacent principals is computed by multiplying the values over link sequences in the obvious way. Assume that the reliability constraint is that principal X can send (resp. receive) a message to (from) principal Y if and only if her trust in Y is not lower than 60%. Principal E can therefore only communicate through principal D. So, assuming that trust values are publicly known, E cannot possibly keep her identity from D as soon as she tries to interact at all. These examples document the existence of an inherent trade-off between anonymity and trust. The fundamental challenge is to achieve an appropriate balance between practical privacy, and acceptable network performance.

Community-based reputation systems are becoming increasingly popular both in the research literature and in practical applications. They are systems designed to estimate the trustworthiness of principals participating in some activity, as well as predict their future behaviour. Metrics for trustworthiness are primarily based on *peer-review*, where peers can rate each other according to the quality they experienced in their past mutual interactions [12, 13, 20]. A good reputation indicates a peer's good past behaviour, and is reflected in a high trust value. Recent research in this domain has raised fundamental issues in the design of reputation management systems for anonymous networks. In particular,

1. what metrics are suitable for computing trust for a given application field?
2. how to ensure the integrity of the peers' trust values, i.e., how to securely store and access trust values against malicious peers?
3. how to ensure that honest users accurately rate other members?

The latter issue requires a mechanism to distinguish a user's bad behaviour resulting from her being under attack, from a deliberately malicious behaviour. This is a challenging and fundamental problem. Indeed, if we cannot accurately tell these two situations apart, malicious users will target honest members in order to deteriorate their performance, and hence reduce other members' trust in them, while maintaining their apparent good behaviour. Thus, honest users may in the long term end up enjoying very low trust levels, while attackers might see their reputation increased, and so they increase their probability of being trusted by others. Over time this will, of course, severely affect the system's anonymity performance. Nevertheless, although a considerable effort has recently been devoted to tackle the first two issues [7, 8, 31], to the best of our knowledge the latter has been so far relatively ignored.

In this paper we investigate the effect of attacks to the trust level of honest users on the security of existing anonymity networks, such as the Reiter and Rubin's Crowds protocol [28] and onion routing networks [10].

The Crowds protocol allows Internet users to perform anonymous web transactions by sending their messages through a random chain of users participating in the protocol. Each user in the '*crowd*' must establish a path between her and a set of servers by selecting randomly some users to act as routers (or forwarders). The formation of such routing paths is performed so as to guarantee that users do not know whether their predecessors are message originators or just forwarders. Each user only has access to messages routed through her. It is well known that Crowds cannot ensure strong anonymity in presence of corrupt participants [5, 28], yet when the number of corrupt users is sufficiently small, it provides a weaker notion of anonymity known as *probable innocence*. Informally, a sender is probably innocent if to an attacker she is no more likely to be the message originator than not to be.

Networks based on Onion Routing are distributed anonymising networks that use *onion routing* [32] to provide anonymity to their users. Similarly to Crowds, users choose randomly a path through the network in which each node knows its predecessor and successor, but no other node. The main difference with respect to Crowds is that traffic flows through the path in cells, which are created by the initiator by successively encrypting the message with the session keys of the nodes in the path, in reverse order. Each node in the act of receiving the message peels the topmost layer, discovers who the next node is, and then relays it forward. In particular, only the last node can see the message in clear and learn its final destination.

In the paper we propose two variants of the congestion attacks in the literature, aimed at deteriorating the trust level of target users in different extension of the Crowds protocol. More specifically, we first extend the protocol so that trust is used to inform the selection of forwarding users. Our analysis of this extension shows that a DoS type attack targeting a user who initially enjoys satisfactory anonymity protection, may threaten her privacy, as her trust level quickly decreases over the time. We then extend the protocol further with a more advanced message forwarding technique, namely onion routing. While this extension offers much better protection than the previous one, our analysis ultimately shows that it suffers from similar DoS attacks as the others.

Related work. Anonymity networks date back thirty years, to when Chaum introduced the concept of *Mix-net* [6] for anonymous communications, where different sources send encrypted messages to a mix which forwards them to their respective destinations. Various designs [1, 10, 15, 24–26, 28, 29, 32] have since been proposed to improve Chaum's mixes, e.g., by combinations of artificial delays, variation in message ordering, encrypted message formats, message batching, and random chaining of multiple mixes.

A variety of attacks [2, 4, 10, 14, 18, 21–23, 27] have since been discovered against such anonymity systems. Those most related to the present work are the so-called *con-*

gestion or *clogging* attacks. In an congestion attack, the adversary monitors the flow through a node, builds paths through other nodes, and tries to use all of their available capacity [2]. The idea is that if the congested node belongs to the monitored path, the variation in the messages' arrival times will reflect at the monitored node. In [23], Murdoch and Danezis describe a congestion attack that may allow them to reveal all Tor's routers (cf. [10]) involved in a path. However, although their attack works well against a Tor network of a relatively small size, it fails against networks of typical sizes, counting nodes in the thousands. More recently, Evans *et al.* [14] improved Murdoch and Danezis's attack so as to practically de-anonymise Tor's users in currently deployed system. A similar attack against MorphMix [29] was recently described by Mclachlan and Hopper [21], proving wrong the previously held view that MorphMix is robust against such attacks [34]. Finally, a congestion attack is used by Hopper *et al.* [18] to estimate the latency between the source of a message and its first relay in Tor. In *loc. cit.* the authors first use a congestion attack to identify the path, and then create a parallel circuit throughout the same path to make their measurements.

Numerous denial of service (DoS) attacks have been reported in the literature. In particular, the *'packet spinning'* attack of [27] tries to lure users into selecting malicious relays by targeting honest users by DoS attacks. The attacker creates long circular paths involving honest users and sends large amount of data through the paths, forcing the users to employ all their bandwidth and then timing out. These attacks motivate the demand for mechanisms to enhance the reliability of anonymity networks. In recent years, a considerable amount of research has been focusing on defining such mechanisms. In particular, trust-and-reputation-based metrics are quite popular in this domain [3, 7–9, 11, 31, 33]. Enhancing the reliability by trust, not only does improve the system's usability, but may also increase its anonymity guarantee. Indeed, a trust-based selection of relays improves both the reliability and the anonymity of the network, by delivering messages through 'trusted' routers. Moreover, the more reliable the system, the more it may attract users and hence improve the anonymity guarantee by growing the anonymity set. Introducing trust in anonymity networks does however open the flank to novel security attacks, as we prove in this paper.

In a recent paper of ours [30] we have analysed the anonymity provided by Crowds extended with some trust information, yet against a completely different threat model. The two papers differ in several ways. Firstly, [30] considers a global and 'credential-based' trust notion, unlike the individual-and-reputation-based trust considered here. Secondly, in [30] we considered an attack scenario where all protocol members are honest but vulnerable to being corrupted by an external attacker. The global and fixed trust in a user contrasts with the local and dynamic trust of this paper, as is meant to reflect the user's degree of resistance against corruption, that is the probability that the external attacker will fail to corrupt her. The paper derives necessary and sufficient conditions to define a 'social' policy of selecting relays nodes in order to achieve a given level of anonymity protection to all members against such attackers, as well as a 'rational' policy maximise one's own privacy.

Structure of the paper. The paper is organised as follows: in §2 we fix some basic notations and recall the fundamental ideas of the Crowds protocol and its properties, including the notion of probable innocence. In §3 we present our first contribution: the

Crowds protocol extended with trust information in the form of a forwarding policy of its participating members, and the privacy properties of the resulting protocol are studied; §4 repeats the analysis for an extension of the protocol with a more advanced forwarding technique inspired by onion routing. Finally, §5 introduces a new 'adaptive' attack scenario, and presents some preliminary results on its analysis, both for the protocol with and without onion forwarding.

2 Crowds

In this section, we briefly revise the Crowds protocol and the notion of probable innocence.

2.1 The Protocol

Crowds is a protocol proposed by Reiter and Rubin in [28] to allow Internet users to perform anonymous web transactions by protecting their identities as originators of messages. The central idea to ensure anonymity is that the originator forwards the message to another, randomly-selected user, which in turn forwards the message to a third user, and so on until the message reaches its destination (the end server). This routing process ensures that, even when a user is detected sending a message, there is a substantial probability that she is simply forwarding it on behalf of somebody else.

More specifically, a crowd consists of a *fixed* number of users participating in the protocol. Some members (users) of the crowd may be corrupt (the *attackers*), and they collaborate in order to discover the originator's identity. The purpose of the protocol is to protect the identity of the message originator from the attackers. When an *originator* –also known as *initiator*– wants to communicate with a server, she creates a random *path* between herself and the server through the crowd by the following process.

- *Initial step:* the initiator selects randomly a member of the crowd (possibly herself) and forwards the request to her. We refer to the latter user as the *forwarder*.
- *Forwarding steps:* a forwarder, upon receiving a request, flips a *biased* coin. With probability $1 - p_f$ she delivers the request to the end server. With probability p_f she selects randomly a new forwarder (possibly herself) and forwards the request to her. The new forwarder repeats the same forwarding process.

The response from the server to the originator follows the same path in the opposite direction. Users (including corrupt users) are assumed to only have access to messages routed through them, so that each user only knows the identities of her immediate predecessor and successor in the path, as well as the server.

2.2 Probable Innocence

Reiter and Rubin have proposed in [28] a hierarchy of anonymity notions in the context of Crowds. These range from '*absolute privacy*,' where the attacker cannot perceive the presence of an actual communication, to '*provably exposed*,' where the attacker can prove a sender-and-receiver relationship. Clearly, as most protocols used in practice, Crowds cannot ensure absolute privacy in presence of attackers or corrupted users, but can only provide weaker notions of anonymity. In particular, in [28] the authors propose an anonymity notion called *probable innocence* and prove that, under some conditions on the protocol parameters, Crowds ensures the probable innocence property to the originator. Informally, they define it as follows:

> *A sender is probably innocent if, from the attacker's point of view, she appears no more likely to be the originator than to not be the originator.* (1)

In other words, the attacker may have reason to suspect the sender of being more likely than any other potential sender to be the originator, but it still appears at least as likely that she is not.

We use capital letters A, B to denote discrete random variables and the corresponding small letters a, b and calligraphic letters $\mathcal{A}$, $\mathcal{B}$ for their values and set of values respectively. We denote by $P(a), P(b)$ the probabilities of a and b respectively and by $P(a, b)$ their *joint probability*. The *conditional probability* of a given b is defined as

$$P(a \mid b) = \frac{P(a, b)}{P(b)} \ .$$

Bayes Theorem relates the conditional probabilities $P(a \mid b)$ and $P(a \mid b)$ as follows

$$P(a \mid b) = \frac{P(b \mid a)\, P(a)}{P(b)} \ . \tag{2}$$

Let n be the number of users participating in the protocol and let c and $n - c$ be the number of the corrupt and honest members, respectively. Since anonymity makes only sense for honest users, we define the set of anonymous events as $\mathcal{A} = \{a_1, a_2, \ldots, a_{n-c}\}$, where a_i indicates that user i is the initiator of the message.

As it is usually the case in the analysis of Crowds, we assume that attackers will always deliver a request to forward immediately to the end server, since forwarding it any further cannot help them learn anything more about the identity of the originator. Thus in any given path, there is at most one detected user: the first honest member to forward the message to a corrupt member. Therefore we define the set of observable events as $\mathcal{O} = \{o_1, o_2, \ldots, o_{n-c}\}$, where o_j indicates that user j forwarded a message to a corrupted user. In this case we also say that user *j is detected* by the attacker.

Reiter and Rubin [28] formalise their notion of probable innocence via the conditional probability that the initiator is detected given that any user is detected at all. This property can be written in our setting as the probability that user i is detected given that she is the initiator, that is the conditional probability $P(o_i \mid a_i)$.[1] Probable innocence holds if

$$\forall i.\ P(o_i \mid a_i) \leq \frac{1}{2} \tag{3}$$

[1] We are only interested in the case in which a user is detected, although for the sake of simplicity we shall not note that condition explicitly.

Reiter and Rubin proved in [28] that, in Crowds, the following holds:

$$P(o_j \,|\, a_i) \;=\; \begin{cases} 1 - \dfrac{n-c-1}{n} p_f & i = j \\[2ex] \dfrac{1}{n} p_f & i \neq j \end{cases} \tag{4}$$

Therefore, probable innocence (3) holds if and only if

$$n \geq \frac{p_f}{p_f - 1/2}(c+1) \quad \mathit{and} \quad p_f \geq \frac{1}{2}$$

As previously noticed in several papers (e.g., [5]), there is a mismatch between the idea of probable innocence expressed informally by (1), and the property actually proved by Reiter and Rubin, viz. (3). The former seems indeed to correspond to the following interpretation given by Halpern and O'Neill [16]:

$$\forall i, j.\; P(a_i \,|\, o_j) \leq \frac{1}{2}\;. \tag{5}$$

In turn, this has been criticised for relying on the probability of users' actions, which the protocol is not really in control of, and for being too strong. However, both (3) and (5) work satisfactorily for Crowds, thanks to its high symmetry: in fact, they coincide under its standard assumption that the *a priori* distribution is uniform, i.e., that each honest user has equal probability of being the initiator, which we follow in this paper too.

We remark that the concept of probable innocence was recently generalised in [17]. Instead of just comparing the probability of being innocent with the probability of being guilty, the paper focusses on the degree of innocence. Formally, given a real number $\alpha \in [0,1]$, a protocol satisfies α-probable innocence if and only if

$$\forall i, j.\; P(a_i \,|\, o_j) \leq \alpha \tag{6}$$

Clearly α-probable innocence coincides with standard probable innocence for $\alpha = 1/2$.

3 Trust in Crowds

In the previous section, we have revised the fundamental ideas of the Crowds protocol and its properties under the assumption that all members are deemed equal. However, as observed in §1, this is clearly not a realistic assumption for today's open and dynamic systems. Indeed, as shown by the so-called 'packet spinning' attack [27], malicious users can attempt to make honest users select bogus routers by causing legitimate routers time out. The use attributes relating to some level of *trust* is therefore pivotal to enhance the reliability of the system. In this section, we firstly reformulate the Crowds protocol under a novel scenario where the interaction between participating users is governed by their level of mutual trust; we then evaluate its privacy guarantees using property (6). We then focus on the analysis of attacks to the trust level of honest users and their impact on the anonymity of the extended protocol. Finally, we investigate the effect of a congestion attack [14] to the trust level of honest users.

3.1 Crowds Extended

We now extend the Crowds protocol to factor in a notion of trust for its participating members. To this end, we associate a trust level t_{ij} to each pair of users i and j, which represents user i's trust in user j. Accordingly, each user i defines her *policy of forwarding* to other members (including herself) based on her trust in each of them. A policy of forwarding for user i is a discrete probability distribution $\{q_{i1}, q_{i2}, \cdots, q_{in}\}$, where q_{ij} denotes the probability that i chooses j as the forwarder, once she has decided to forward the message.

A natural extension of Crowds would obviously allow the initiator to select her first forwarder according to her own policy, and then leave it to the forwarder to pick the next relay, according to the forwarder's policy. This would however have the counterintuitive property that users may take part in the path which are not trusted by the initiator, just because they are trusted by a subsequent forwarder. We rather take the same view as most current systems, that the initiator is in charge of selecting the entire path which will carry her transactions. When an initiator wants to communicate with a server, she selects a random *path* through the crowd between herself and the server by the following process.

- *First forwarder:* with probability q_{ij} the initiator i selects a member j of the crowd (possibly herself) according to her policy of forwarding $\{q_{i1}, q_{i2}, \cdots, q_{in}\}$.
- *Subsequent forwarders:* the initiator flips a *biased* coin; with probability $1 - p_f$ the current forwarder will be the last on the path, referred to as the *path's exit user*. Otherwise, with probability $p_f \times q_{ik}$, she selects k (possibly herself) as the next forwarder in the path; and so on until a path's exit user is reached.

The initiator then creates the path iteratively as follows. She establishes a session key by performing an authenticated key exchange protocol, such as Diffie-Hellman,[2] with the first forwarder F_1. At each of subsequent iteration $i \geq 2$, the initiator uses the partially-formed path to send F_{i-1} an encrypted key exchange message to be relayed to F_i. In this way, the path is extended to F_i, and the use of session keys guarantees that any intermediary router only knows her immediate predecessor and successor. Once the path is formed, messages from the initiator to the server are sent in the same way as in the normal Crowds. Thus, all the nodes in the path have access to the contain of the message and, obviously, to the end server. In particular, this means that the notion of detection remains the same in the extended protocol as in the original one.

Then we use our probabilistic framework to evaluate Crowds extended protocol. We start by evaluating the conditional probability $P(o_j \mid a_i)$. Let η_i (resp. $\zeta_i = 1 - \eta_i$) be the overall probability that user i chooses a honest (resp. corrupt) member as a forwarder. Then we have the following result.

Proposition 1

$$P\left(o_j \mid a_i\right) \;=\; \zeta_i \epsilon_{ij} + \frac{q_{ij}\zeta_i p_f}{1 - \eta_i p_f}\,,$$

where $\eta_i = \sum_{k \leq (n-c)} q_{ik}$, $\zeta_i = \sum_{k \leq c} q_{ik}$ *and* $\epsilon_{ij} \;=\; \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$

[2] We assume that public keys of participating users are known.

Proof. Let k denote the position occupied by the first honest user preceding an attacker on the path, with the initiator occupying position zero. Let $P(o_j \mid a_i)_{(k)}$ denote the probability that user j is detected exactly at position k. Only the initiator can be detected at position zero, and the probability that this happens is equal to the overall probability that the initiator chooses a corrupt member as a forwarder. Therefore

$$P\left(o_j \mid a_i\right)_{(0)} = \begin{cases} \zeta_i & i = j \\ 0 & i \neq j \end{cases}$$

Now the probability that j is detected at position $k > 0$ is given by

- the probability that she decides to forward k times and picks $k - 1$ honest users, i.e.,$p_f^{k-1}\eta_i^{k-1}$ (recall that at the initial step she does not flip the coin),
- times the probability of choosing j as the kth forwarder, i.e., q_{ij},
- times the probability that she picks any attacker at stage $k + 1$, i.e., $\zeta_i p_f$.

Therefore

$$\forall k \geq 1, \;\; P\left(o_j \mid a_i\right)_{(k)} = \eta_i^{k-1} p_f^k q_{ij} \zeta_i$$

and hence

$$\begin{aligned} P\left(o_j \mid a_i\right) &= \sum_{k=0}^{\infty} P\left(o_j \mid a_i\right)_{(k)} \\ &= \zeta_i \epsilon_{ij} + \sum_{k=1}^{\infty} \eta_i^{k-1} p_f^k q_{ij} \zeta_i \\ &= \zeta_i \epsilon_{ij} + \sum_{k=0}^{\infty} \eta_i^{k} p_f^{k+1} q_{ij} \zeta_i \\ &= \zeta_i \epsilon_{ij} + p_f q_{ij} \zeta_i \sum_{k=0}^{\infty} \eta_i^k p_f^k \\ &= \zeta_i \epsilon_{ij} + \frac{q_{ij} \zeta_i p_f}{1 - \eta_i p_f} \, . \end{aligned}$$

An immediate consequence is that when user i initiates a transaction, user j is not detectable if and only if the initiator's policy of forwarding never chooses an attacker or j as forwarder.

Corollary 1. $P(o_j \mid a_i) = 0$ *if and only if one of the following holds:*

1. $\zeta_i = 0$;
2. $q_{ij} = 0$ *and* $i \neq j$.

Now, let us compute the probability of detecting a user $P(o_j)$. We assume a uniform distribution for anonymous events.

Proposition 2. *If the honest members are equally likely to initiate a transaction, then*

$$P(o_j) = \frac{1}{n-c}\left(\zeta_j + \sum_{i \le (n-c)} \frac{q_{ij}\zeta_i p_f}{1 - \eta_i p_f}\right),$$

where ζ_j and η_i are defined as in Proposition 1.

Proof. Since the anonymous events are uniformly distributed then $P(a_i) = 1/(n-c)$ for all i. Thus

$$\begin{aligned} P(o_j) &= \sum_{i \le (n-c)} P\left(o_j \mid a_i\right) P(a_i) \\ &= \sum_{i \le (n-c)} P\left(o_j \mid a_i\right) \frac{1}{n-c} \\ &= \frac{1}{n-c} \sum_{i \le (n-c)} P\left(o_j \mid a_i\right) \\ &= \frac{1}{n-c} \sum_{i \le (n-c)} \left(\zeta_i \epsilon_{ij} + \frac{q_{ij}\zeta_i p_f}{1 - \eta_i p_f}\right) \\ &= \frac{1}{n-c}\left(\zeta_j + \sum_{i \le (n-c)} \frac{q_{ij}\zeta_i p_f}{1 - \eta_i p_f}\right). \end{aligned}$$

As one could expect, a user j is not detectable if both herself and any user i that might include j in her path never choose a corrupted member as a forwarder. Formally:

Corollary 2. $P(o_j) = 0$ *if and only if*

$$\zeta_j = 0 \quad \textit{and} \quad \forall i.\,(\, q_{ij} = 0 \;\textit{or}\; \zeta_i = 0\,)\,.$$

Now from Propositions 1 and 2 and Bayes Theorem (2), we have the following expression for the degree of anonymity provided by the extended protocol, which holds when $P(o_j) \neq 0$.

Proposition 3. *If the honest members are equally likely to initiate a transaction, then*

$$P\left(a_i \mid o_j\right) = \frac{\zeta_i \epsilon_{ij} + \dfrac{q_{ij}\zeta_i p_f}{1 - \eta_i p_f}}{\zeta_j + \displaystyle\sum_{k \le (n-c)} \frac{q_{kj}\zeta_k p_f}{1 - \eta_k p_f}},$$

where ζ_i and η_j are defined as above.

It is now easy to see that if all honest users have uniform probability distributions as forwarding policies, the extended protocol reduces to the original Crowds protocol.

Corollary 3. *If for all i and j, $q_{ij} = 1/n$, then $\eta_i = (n-c)/n$ and $\zeta_i = c/n$. Therefore*

$$P\big(a_i \mid o_j\big) = \begin{cases} 1 - \dfrac{n-c-1}{n} p_f & i = j \\ \dfrac{1}{n} p_f & i \neq j \end{cases}$$

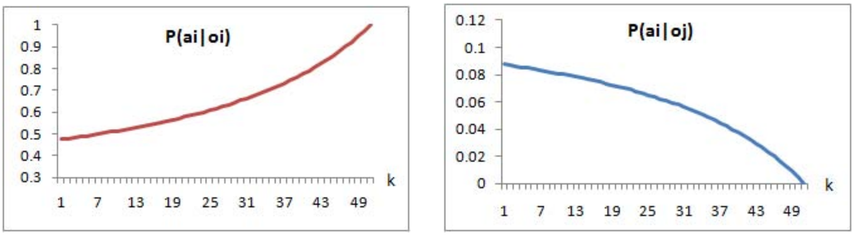
3.2 On the Security of Extended Crowds

Here we show that the absence of a uniform forwarding policy makes it very hard to achieve probable innocence as defined by Halpern and O'Neill ([5]). Indeed consider the following instance of the protocol, where three honest users $\{1, 2, 3\}$ face a single attacker $\{4\}$. Assume that the honest users are aware of the malicious behaviour of 4, and choose their forwarding policies as follows: $p_f = 2/3$, and $q_{1j} = q_{2j} = 1/3$, and $q_{3j} = 0.33$ for all $j \leq 3$. In other words, the first two choose uniformly any honest users as a forwarder and never pick the attacker, whilst the third one may choose the attacker, though with a small probability $q_{34} = 0.01$. Thus, $\zeta_1 = \zeta_2 = q_{14} = q_{24} = 0$ and $\zeta_3 = q_{34} = 0.01$. It follows that $P(a_3 \,|\, o_j) = 1$, for all j, and the instance does not ensure probable innocence, even though the third user's policy is after all very similar to those of the other honest users. This is because if someone is detected, then user 3 is necessarily the initiator, as she is the only one who might possibly pick the attacker in her path.

Observe however that this instance of the protocol ensures probable innocence in Reiter and Rubin's formulation: indeed, for all honest users i and j, $P(o_j \,|\, a_i) < 0.0165$. The key difference at play here is that Halpern and O'Neill's definition is stronger, as it focuses on the probability that a specific user is the initiator once somebody has been detected, regardless of the probability of the detection event. On the other hand, Reiter and Rubin's formula measures exactly (the conditional probability of) the latter. This means that if the probability of detection is small, as in this case, systems may be classified as statistically secure even when one such detection event may lead to complete exposure for some initiators, as in this case.

Attackings trust. As already observed by its authors, Crowds is vulnerable to denial of service (DoS) attacks: it is enough that a single malicious router delays her forwarding action to severely hinder the viability of an entire path. This kind of attack is in fact hard for the initiator to respond to. Just because the creation of multiple paths by any single user substantially increases their security risk, the initiator has a strong incentive to keep using the degraded path. Indeed, it is advisable in Crowds to modify a path only when it has collapsed irremediably, e.g. due to a system crash of a router, or their quitting the crowd. In this case the path is re-routed from the node preceding the failed router. As a consequence, recent research has been devoted to developing '*trust metrics*' meant enhance the reliability of anonymity systems [7, 8, 31].

Although the primary goal of incorporating trust in anonymity networks is to 'enhance' the privacy guarantees by routing messages through *trusted* relays, preventing the presence of attackers in forwarding paths is in itself not sufficient. External attackers



(a) $i = j = 7$

(b) $i \neq j = 7$

Fig. 2. Crowds extended

may in fact target honest users with DoS attacks independent of the protocol, to make them look unreliable and/or unstable. In this way, the target users will gradually loose others members' trust, whilst internal attackers may keep accruing good reputations. Thus, over the time the trust mechanisms may become counterproductive.

Let us illustrate an attack of this kind. Consider an instance of the protocol where seven honest users $\{1, 2, \cdots, 7\}$ face a single attacker $\{8\}$, assume that 7 is the honest user targeted by the attack, and that all users are equally likely to initiate a transaction. Recall that a path in Crowds remains fixed for a certain amount of time –typically one day– known as a *session*. In practice, *all* transactions initiated by a given user follow the same path, regardless of their destination servers. At the end of the session then, all existing paths are destroyed, new members can join the crowd, and each member willing to initiate anonymous transactions creates a new path. Trust level updates play therefore their role at the beginning of each session. For the purpose of this example, we assume that the protocol is equipped with mechanisms to detect unstable routers (e.g., by monitoring loss of messages, timeouts, variations in response time and so on); upon realising that her path is unstable, an initiator will notify all members of the identity of the unstable node (in this case 7).[3] When a node is reported as unstable, all other honest nodes decrease their trust in her at the beginning of the following session. For simplicity, we assume that all users start with the same trust level τ, and that the target user remains fixed over time. The following policies of forwarding are therefore in place for each session, with $n = 8$, $c = 1$ and $\tau = 50$.

$$q_{ij}^{(k)} = \begin{cases} \dfrac{1}{n} & i = 7 \\[2ex] \dfrac{\tau - k}{n \times \tau - k} & i \neq 7 \text{ and } j = 7 \\[2ex] \dfrac{\tau}{n \times \tau - k} & i \neq 7 \text{ and } j \neq 7\,. \end{cases}$$

In words, honest users other that the target decrease their trust in her by one and re-distributed it uniformly to the remaining users. On the other hand, the target has no

[3] This contrasts with the approach of [11], where the initiator would directly decrease her trust in all users in the path.

reason to change her trust, as there is no evidence to suspect anybody as the source of the external attack. Thus, her policy remains the same over the time. Hence, we have

$$\zeta_i^{(k)} = \begin{cases} \dfrac{c}{n} & i = 7 \\[2ex] \dfrac{\tau}{n \times \tau - k} & \text{otherwise.} \end{cases}$$

Assuming that the forwarding probability is $p_f = 0.7$, Figure 2 shows the probability that the target will be identified over time. Clearly, the target's privacy deteriorates quickly, as it becomes increasingly unlikely that users other than herself pick her when building a path. In particular, after seven sessions the protocol can no longer ensure probable innocence as the probability $P(a_7 \mid o_7)$ becomes greater than 0.5.

4 Onion Forwarding in Crowds

In the previous section we analysed the privacy protection afforded by Crowds extended with a notion of trust. Following a similar pattern, in this section we focus on the privacy guarantees offered by our protocol when equipped with '*onion forwarding*,' a superior forwarding technique used in systems actually deployed, such as Tor [10].

In Crowds, any user participating in a path has access to the cleartext messages routed through it. In particular, as all relay requests expose the message's final destination, a team of attackers will soon build up a host of observations suitable to classify the behaviour of honest participants. We recently proved in [17] that such extra attackers' knowledge makes it very difficult to achieve anonymity in Crowds. The most effective technique available against such a risk is onion forwarding, originally used in the 'Onion Routing' protocol [32], and currently implemented widely in real-world systems. The idea is roughly as follows. When forming a path, the initiator establishes a set of session *encryption keys*, one for each user in it, which she then uses to repeatedly encrypt each message she routes through, starting with the last node on the path, and ending with the first. Each intermediate user, in the act of receiving the message decrypts it with her key. Doing so, she 'peels' away the outmost layer of encryption, discovers who the next forwarder is, and relays the message as required. In particular, only the last node sees the message in clear and learns its actual destination. Thus, a transaction is detected only if the last user in the path, also known as the '*exit node*,' is an attacker, and the last honest user in the path is then detected.

4.1 Privacy Level of the Onion Forwarding

Next we study the privacy ensured to each member participating in the protocol under the onion forwarding scheme. As we did earlier, we begin with computing the conditional probability $P(o_j \mid a_i)$.

Proposition 4

$$P\left(o_j \mid a_i\right) = (1 - p_f)\, \zeta_i \epsilon_{ij} + \frac{q_{ij}\, \zeta_i p_f}{1 - \zeta_i p_f}\ .$$

Proof. Let k denote the last position occupied by an honest user preceding an attacker on the path, i.e., the position of the detected user. We denote by $P(o_j \,|\, a_i)_{(k)}$ the probability that user j is detected exactly at position k. Again, only the initiator can be detected at position zero, and the probability that this happens is equal to the overall probability that the initiator chooses a corrupt member as a forwarder, multiplied by the probability that the latter is the last node in the path. Therefore

$$P\left(o_j \mid a_i\right)_{(0)} = \begin{cases} (1-p_f)\,\zeta_i & i = j \\ 0 & i \neq j \end{cases}$$

Now the probability that j is detected at position $k > 0$ is given by

- the probability that she decides to forward k times and picks $k-1$ users (does not matter whether honest or not, as non-exit attackers cannot see the messages), i.e., p_f^{k-1} (recall that at the initial step she does not flip the coin),
- times the probability of choosing j as the kth forwarder, i.e. q_{ij},
- times the probability that she picks any number k' of attackers at the end of the path, i.e. $\sum_{k'=1}^{\infty} p_f^{k'}\,\zeta_i^{k'}\,(1-p_f)$.

Therefore

$$\forall k \geq 1,\;\; P\left(o_j \mid a_i\right)_{(k)} = \sum_{k=1}^{\infty}\left(p_f^{k-1} q_{ij} \sum_{k'=1}^{\infty} p_f^{k'} \zeta_i^{k'} (1-p_f)\right),$$

and hence

$$\begin{aligned}
P\left(o_j \mid a_i\right) &= \sum_{k=0}^{\infty} P\left(o_j \mid a_i\right)_{(k)} \\
&= (1-p_f)\zeta_i\epsilon_{ij} + \sum_{k=1}^{\infty}\left(p_f^{k-1} q_{ij} \sum_{k'=1}^{\infty} p_f^{k'} \zeta_i^{k'} (1-p_f)\right) \\
&= (1-p_f)\left[\zeta_i\epsilon_{ij} + q_{ij}\sum_{k=1}^{\infty}\left(p_f^{k-1}\sum_{k'=1}^{\infty} p_f^{k'}\zeta_i^{k'}\right)\right] \\
&= (1-p_f)\left[\zeta_i\epsilon_{ij} + q_{ij}\sum_{k=1}^{\infty} p_f^{k-1}\frac{\zeta_i p_f}{1-\zeta_i p_f}\right] \\
&= (1-p_f)\left[\zeta_i\epsilon_{ij} + \frac{q_{ij}\zeta_i p_f}{1-\zeta_i p_f}\,\frac{1}{1-p_k}\right] \\
&= (1-p_f)\left[\zeta_i\epsilon_{ij} + \frac{q_{ij}\zeta_i p_f}{(1-p_f)(1-\zeta_i p_f)}\right].
\end{aligned}$$

Corollary 4. $P(o_j \,|\, a_i) = 0$ *if and only if one of the following holds:*

1. $\zeta_i = 0\,;$
2. $q_{ij} = 0$ *and* $i \neq j$.

Now on the probability of detecting a user $P(o_j)$. Assuming uniform distribution of anonymous events we have the following result.

Proposition 5. *If the honest member are equally likely to initiate a transaction then.*

$$P(o_j) = \frac{1}{n-c}\left((1-p_f)\,\zeta_j + \sum_{i\leq(n-c)} \frac{q_{ij}\,\zeta_i p_f}{1-\zeta_i p_f}\right).$$

Proof. Since the anonymous events are uniformly distributed then $P(a_i) = 1/(n-c)$ for all i. Thus

$$\begin{aligned}
P(o_j) &= \sum_{i\leq(n-c)} P\left(o_j \mid a_i\right) P(a_i)\\
&= \sum_{i\leq(n-c)} P\left(o_j \mid a_i\right) \frac{1}{n-c}\\
&= \frac{1}{n-c} \sum_{i\leq(n-c)} P\left(o_j \mid a_i\right)\\
&= \frac{1}{n-c} \sum_{i\leq(n-c)} \left((1-p_f)\zeta_i\epsilon_{ij} + \frac{q_{ij}\zeta_i p_f}{1-\zeta_i p_f}\right)\\
&= \frac{1}{n-c}\left((1-p_f)\zeta_j + \sum_{i\leq(n-c)} \frac{q_{ij}\zeta_i p_f}{1-\zeta_i p_f}\right).
\end{aligned}$$

We then have the same conditions of non-detectability as in the previous section; that is, the following result holds.

Corollary 5. $P(o_j) = 0$ *if and only if*

$$\zeta_j = 0 \quad \textit{and} \quad \forall i.\,(\,q_{ij} = 0 \textit{ or } \zeta_i = 0\,)\,.$$

Now from Proposition 4 and 5 and the Bayes theorem, we have the following result.

Proposition 6. *If the honest members are equally likely to initiate a transaction, then*

$$P\left(a_i \mid o_j\right) = \frac{\zeta_i\epsilon_{ij} + \dfrac{q_{ij}\,\zeta_i\,p_f}{(1-p_f)(1-\zeta_i\,p_f)}}{\zeta_j + \displaystyle\sum_{k\leq(n-c)} \frac{q_{kj}\,\zeta_k\,p_f}{(1-p_f)(1-\zeta_k\,p_f)}}\,.$$

Now from Propositions 3 and 6, we can prove effectively that the privacy level ensured by the onion version is better than those offered by the versions where messages are forwarded in cleartext. More formally, let $\left[P(a_i \mid o_j)\right]_{CR}$ and $\left[P(a_i \mid o_j)\right]_{OR}$ denote the probability that i is the initiator given that j is detected under cleartext routing and onion routing, respectively. Then the following holds.

Theorem 1. $\left[\,P(a_i \mid o_j)\,\right]_{OR} \leq \left[\,P(a_i \mid o_j)\,\right]_{CR}$.

4.2 On the Security of the Onion Forwarding Version

As mentioned before, onion forwarding is the forwarding technique of choice in several real-world systems. Recent work [14, 18, 21–23] shows that such systems are vulnerable to so-called *congestion attacks*, which intuitively work as follows. Assume that the initiator selects a path which contains a corrupt user as the exit node. The attacker can then observe the pattern of arrival times of the initiator's requests, and tries to identify the entire path by selectively congesting the nodes she suspect to belong to it. Precisely, to determine whether or not a specific node occurs in the path, she asks a collaborating attacker to build a long path looping on the target node and ending with a corrupt node. Using this, the attacker perturbs the flow through the target node, so that if the latter belongs also to the path under observation, the perturbation will reflect at its exit node.

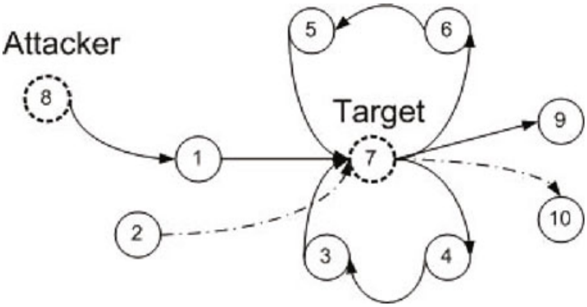


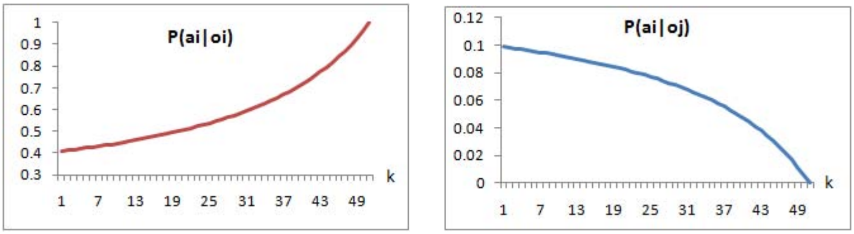
Fig. 3. Congestion attack

Here we use a variant of the congestion attack which, similarly to the previous section, allows internal attackers to deteriorate the reputation of a targeted honest user, and does not require the attacker to belong to a path. Figure 3 illustrates the attack, where a long path is built looping as many times as possible over the target, preferably using different loops involving different users. Thank to such properties, the target user will be significantly busy handling the same message again and again, whilst no other member of the path will be congested.

Figure 4 illustrates the effect of this attack using the same example as in the cleartext forwarding version in §3. The results are completely in tune with those presented by Figure 2: even though the target node initially enjoys a better anonymity protection, her anonymity will unequivocally fall, although more smoothly than in §3. In particular, after twenty sessions, the protocol no longer ensures probable innocence, as the probability of identifying the target node becomes greater than 0.5.

5 Adaptive Attackers

We have worked so far under the assumption that protocol participants either behave always honestly or always maliciously. Arguably, this is a rather unrealistic hypothesis



(a) $i = j = 7$

(b) $i \neq j = 7$

Fig. 4. Onion forwarding

in open and dynamic systems, where honest nodes can become malicious upon being successfully attacked. In this section we take the more realistic view that nodes may become corrupt, and study a new kind of attackers, which we dub ‘adaptive,’ and the relative attacks.

Adaptive attackers differ from those we considered so far in the paper –and indeed from those considered so far in the literature on Crowds– in that when they intercept a message, rather than just reporting its sender as the initiator, they attempt to travel the path back in order to improve their chance to catch the actual originator. They do so by trying to *corrupt* the sender of the message, say j_1. If the attack succeeds, then the attacker effectively learns from j_1 all she needs to identify j_1’s predecessor on the path, say j_2, and repeat the adaptive attack on j_2, having moved a step closer to the initiator. The process is repeated iteratively until the attacker either fails to corrupt the current node (or timeouts whilst trying to) or reaches the beginning of the path. When that happens, the attacker reports the current node, say j_k, which is obviously a better candidate than j_1 to have originated the transaction.

We regard this as a significant and realistic kind of attack, as there clearly are a multitude of ways in which the adaptive attacker may attempt to corrupt a node. These range from brute force attacks via virus and worms which gains the attacker complete control over the node, to milder approaches based on luring the target to give away some bit of information in exchange for some form of benefit, and in general are entirely independent of the Crowds protocol. We therefore do not postulate here about the means which may be available to the attacker to carry out her task, make no assumptions whatsoever about her power, and take the simplified view that each node has at all time the same probability π to become corrupted.

In the rest of the section we re-evaluate the privacy guarantees afforded by Crowds extended –with and without onion forwarding– under this new adaptive attack scenario. We shall however carry out the analysis under the unrealistic assumption that it is necessary for attackers to corrupt a node each time they meet her on the path. Recall in fact that a single node will typically appear several times in a path. Therefore, an adaptive attacker in her attempt to travel the path backwards towards the initiator will in general meet the each node several times. The reason why our assumption may be justified is when the attacks only gain the attacker access to just enough data to get to the node’s last predecessor on the path, rather than to the entire set of them. On the other hand, the reason why this assumption is ultimately unsatisfactory is that it is overly dangerous to make limiting assumptions as to the degree of success of an attack, and assess speculatively the extent to which a node's integrity is compromised, the methodologically correct attitude being to assume that the attacker has gained total control over the target. And when she has, by definition she simply has no need to corrupt the node again, and no new knowledge may be acquired by doing so. In the concluding section, we discuss a few preliminary ideas on how to remove this restriction in future work.

5.1 Crowds Extended

Our technical development proceeds *mutatis mutandis* as in §3 and §4. In particular, as before we first evaluate the conditional probability $P(o_j \,|\, a_i)$, then under the hypothesis that all honest users are equally likely to initiate a transaction, we compute $P(o_j)$, and finally, using Bayes Theorem, we obtain $P(a_i \,|\, o_j)$. In this section we omit all proofs.

The probabilities $P(o_i \,|\, a_i)_{(0)}$ and $P(o_j \,|\, a_i)_{(1+)}$ that node i is detected at the initiator position or at any position after that can be expressed respectively as

$$P\Big(o_i \,|\, a_i\Big)_{(0)} = \zeta_i + \frac{p_f\,\eta_i\,\zeta_i\,\pi}{1-\pi}\left(\frac{1}{1-p_f\eta_i} - \frac{\pi}{1-\pi p_f\,\eta_i}\right),$$

$$P\Big(o_j \,|\, a_i\Big)_{(1+)} = \frac{q_{ij}\,\zeta_i\,p_f}{1-p_f\,\eta_i} - \frac{q_{ij}\,\zeta_i\,\eta_i\,p_f^2\,\pi^2}{(1-\pi)(1-p_f\,\zeta_i)}\,,$$

which gives the following result, where again $\epsilon_{ij} = 1$ if $i = j$, and 0 otherwise.

The key to these formulae is to consider that when a user is detected at position h, this is potentially due to a detection at position $h+k$, for any $k \geq 0$, whereby the attacker has successively travelled back k positions on the path, by either corrupting honest users with probability π or by meeting other attackers. The situation would be quite different were we to take into account that the attacker only needs to corrupt a honest user once, as π would not anymore be a constant.

Proposition 7

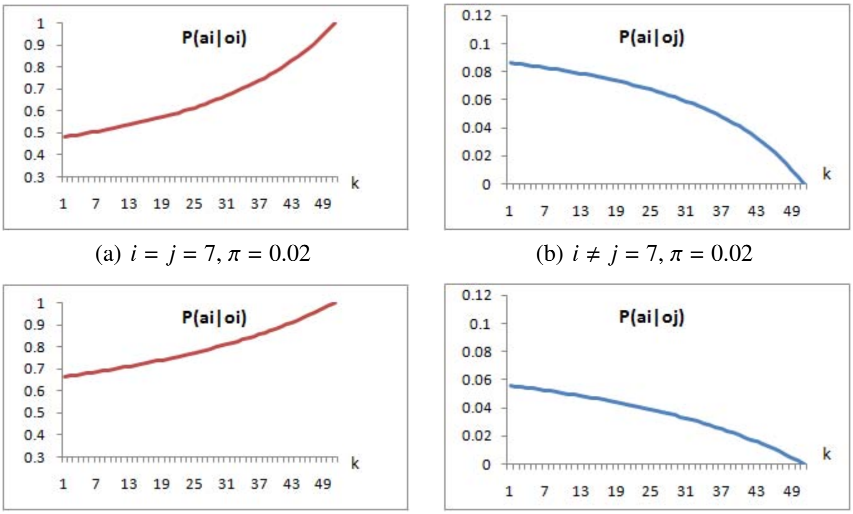
$$P(o_j \,|\, a_i) = \epsilon_{ij} P\Big(o_i \,|\, a_i\Big)_{(0)} + P\Big(o_j \,|\, a_i\Big)_{(1+)}\,.$$

Under the hypothesis of a uniform distribution of anonymous events, it is easy to prove the following.

Proposition 8. *If the honest members are equally likely to initiate a transaction, then*

$$P(o_j) = \frac{1}{n-c}\left(P\Big(o_j \,|\, a_j\Big)_{(0)} + \sum_{k\leq(n-c)} P\Big(o_j \,|\, a_k\Big)_{(1+)}\right).$$

Now from Proposition 7 and 8 and Bayes Theorem, we have the following.



(a) $i = j = 7, \pi = 0.02$

(b) $i \neq j = 7, \pi = 0.02$

(c) $i = j = 7, \pi = 0.5$

(d) $i \neq j = 7, \pi = 0.5$

Fig. 5. Example in Crowds extended against adaptive attack

Proposition 9. *If the honest members are equally likely to initiate a transaction, then*

$$P\big(a_i \,|\, o_j\big) = \frac{\epsilon_{ij} P\big(o_i \,|\, a_i\big)_{(0)} + P\big(o_j \,|\, a_i\big)_{(1+)}}{P\left(o_j \,|\, a_j\right)_{(0)} + \sum_{k \leq (n-c)} P\left(o_j \,|\, a_k\right)_{(1+)}} .$$

Of course, in case the attacker's attempts to travel back the path never succeed, the formula reduces to the one we found previously.

Corollary 6. *If* $\pi = 0$, *that is the attacker is not adaptive, then*

$$P\big(a_i \,|\, o_j\big) = \frac{\zeta_i \epsilon_{ij} + \dfrac{q_{ij}\zeta_i p_f}{1 - \eta_i p_f}}{\zeta_j + \displaystyle\sum_{k \leq (n-c)} \frac{q_{kj}\zeta_k p_f}{1 - \eta_k p_f}} ,$$

which is the same as Proposition 3.

Figure 5 illustrates the formulae $P(a_7 \,|\, o_7)$ and $P(a_i \,|\, o_7)$ for $i \neq 7$ on our running example, where we add $\pi = 0.02$ and $\pi = 0.5$ to the existing parameters, viz., $n = 8$, $c = 1$, $p_f = 0.7$, and $\tau = 50$. It is interesting here to observe the effect of the attacker's corruption power, insofar as that is represented by π: the larger π, the more lethal the attacker, the farther away the protocol from the standard, and the more insecure. In particular, for $\pi = 0.5$ the system fails by a large margin to guarantee probable innocence even before the attack to 7's trust level starts.

5.2 Onion Forwarding

Under onion forwarding, the adaptive attackers must appear as the last node on the path, and from there, starting with her predecessor, try to corrupt nodes back towards the originator. Following the same proof strategy as before, we define obtain the following formulae.

$$P\Big(o_i\,|\,a_i\Big)_{(0)} = (1-p_f)\zeta_i + \frac{p_f\,\eta_i\,\zeta_i\,\pi\,(1-p_f)}{(1-p_f\,\zeta_i)(1-\pi)}\left(\frac{1}{1-\eta_i\,p_f} - \frac{\pi}{1-\pi\,\eta_i p_f}\right),$$

$$P\Big(o_j\,|\,a_i\Big)_{(1+)} = \frac{q_{ij}\,\zeta_i\,p_f}{1-\zeta_i\,p_f} + \frac{p_f^2\,\eta_i\,\zeta_i\,\pi\,q_{ij}}{(1-p_f\,\zeta_i)(1-\pi)}\left(\frac{1}{1-\eta_i\,p_f} - \frac{\pi}{1-\pi\,\eta_i\,p_f}\right),$$

and therefore:

Proposition 10

$$P\Big(o_j\,|\,a_i\Big) = \epsilon_{ij}P\Big(o_i\,|\,a_i\Big)_{(0)} + P\Big(o_j\,|\,a_i\Big)_{(1+)}\,.$$

Now on the probability of detecting a user $P(o_j)$.

Proposition 11. *If the honest members are equally likely to initiate a transaction, then*

$$P(o_j) = \frac{1}{n-c}\left(P\Big(o_j\,|\,a_j\Big)_{(0)} + \sum_{k\leq(n-c)} P\Big(o_j\,|\,a_k\Big)_{(1+)}\right).$$

As before, the result below follows from Propositions 10 and 11 and Bayes Theorem.

Proposition 12. *If the honest members are equally likely to initiate a transaction then.*

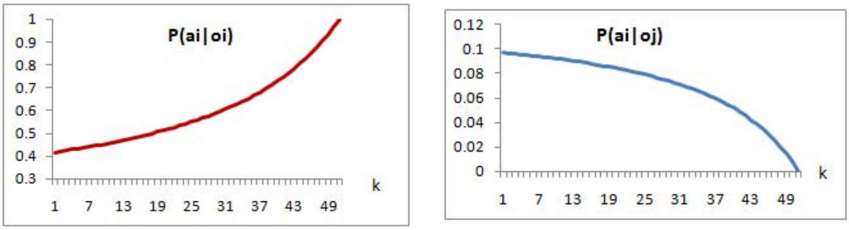
$$P\Big(a_i\,|\,o_j\Big) = \frac{\epsilon_{ij}P\Big(o_i\,|\,a_i\Big)_{(0)} + P\Big(o_j\,|\,a_i\Big)_{(1+)}}{P\Big(o_j\,|\,a_j\Big)_{(0)} + \sum_{k\leq(n-c)} P\Big(o_j\,|\,a_k\Big)_{(1+)}}\,.$$

Corollary 7. *If $\pi = 0$, that is the attacker after all not adaptive, then*

$$P(a_i\,|\,o_j) = \frac{\zeta_i\epsilon_{ij} + \dfrac{q_{ij}\zeta_i p_f}{(1-p_f)(1-\zeta_i p_f)}}{\zeta_j + \displaystyle\sum_{k\leq(n-c)} \frac{q_{kj}\zeta_k p_f}{(1-p_f)(1-\zeta_k p_f)}}\,,$$

which coincides with Proposition 6.

Finally, Figure 6 illustrates $P(a_7\,|\,o_7)$ and $P(a_i\,|\,o_7)$ for $i \neq 7$ on our running example, for $\pi = 0.5$. Although the graphs are shaped as in the previous cases, it is possible to notice the increase security afforded by the onion forwarding.



(a) $i = j = 7$

(b) $i \neq j = 7$

Fig. 6. Onion forwarding against adaptive attacks

6 Conclusion

In this paper we have presented an enhancement of the Crowds anonymity protocol via a notion of trust which allows crowd members to route their traffic according to their perceived degree of trustworthiness of other members. We formalised the idea quite simply by means of (variable) forwarding policies, with and without onion forwarding techniques. Our protocol variation has the potential of improving the overall trustworthiness of data exchanges in anonymity networks, which may naturally not be taken for granted in a context where users are actively trying to conceal their identities. We then analysed the privacy properties of the protocol quantitatively, both for Crowds and onion forwarding, under standard and adaptive attacks.

Our analysis in the case of adaptive attacks is incomplete, in that it assumes that attackers whilst attempting to travel back over a path towards its originator, need to corrupt each honest node each time they meet her. Arguably, this is not so. Typically a node j will act according to a routing table, say T_j. This will contain for each path's id a translation id and a forwarding address (either another user, or the destination server) and, in the case of onion forwarding, the relevant encryption key. (Observe that since path's id are translated at each step, j may not be able to tell whether or not two entries in T_j actually correspond to a same path and, therefore, may not know how many times she occurs on each path.) It is reasonable to assume that upon corruption an attacker c will seize T_j, so that if she ever reaches j again, c will find all the information to continue the attack just by inspecting T_j.

Observe now that the exact sequence of users in the path is largely irrelevant to compute $P(o_j \,|\, a_i)$. It only matters how many times each of them appears in between the attacker at the end of the path and the detected node. Using some combinatorics, it is therefore relatively easy to write a series for $P(o_j \,|\, a_i)$ based on summing up a weighted probability for all possible occurrence patterns of $n - c$ honest users and c attackers in the path. Quite a different story is to simplify that series to distill a usable formula. That is a significant task which we leave for future work.

Acknowledgements. We thank Ehab ElSalamouny and Catuscia Palamidessi for their insights and for proofreading.

References

1. Abe, M.: Universally verifiable Mix-net with verification work indendent of the number of Mix-servers. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 437–447. Springer, Heidelberg (1998)
2. Back, A., Möller, U., Stiglic, A.: Traffic analysis attacks and trade-offs in anonymity providing systems. In: Moskowitz, I.S. (ed.) IH 2001. LNCS, vol. 2137, pp. 245–257. Springer, Heidelberg (2001)
3. Backes, M., Lorenz, S., Maffei, M., Pecina, K.: Anonymous webs of trust. In: 10th Privacy Enhancing Technologies Symposium, PETS 2010. LNCS. Springer, Heidelberg (to appear, 2010)
4. Borisov, N., Danezis, G., Mittal, P., Tabriz, P.: Denial of service or denial of security? In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) ACM Conference on Computer and Communications Security, pp. 92–102. ACM, New York (2007)
5. Chatzikokolakis, K., Palamidessi, C.: Probable innocence revisited. Theor. Comput. Sci. 367(1-2), 123–138 (2006)
6. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM 24(2), 84–88 (1981)
7. Damiani, E., di Vimercati, S.D.C., Paraboschi, S., Pesenti, M., Samarati, P., Zara, S.: Fuzzy logic techniques for reputation management in anonymous peer-to-peer systems. In: Wagenknecht, M., Hampel, R. (eds.) Proceedings of the 3rd Conference of the European Society for Fuzzy Logic and Technology, pp. 43–48 (2003)
8. Damiani, E., di Vimercati, S.D.C., Paraboschi, S., Samarati, P., Violante, F.: A reputation-based approach for choosing reliable resources in peer-to-peer networks. In: Atluri, V. (ed.) ACM Conference on Computer and Communications Security, pp. 207–216. ACM, New York (2002)
9. Dingledine, R., Freedman, M.J., Hopwood, D., Molnar, D.: A reputation system to increase mix-net reliability. In: Moskowitz, I.S. (ed.) IH 2001. LNCS, vol. 2137, pp. 126–141. Springer, Heidelberg (2001)
10. Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: The second-generation onion router. In: USENIX Security Symposium, pp. 303–320. USENIX (2004)
11. Dingledine, R., Syverson, P.F.: Reliable MIX cascade networks through reputation. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 253–268. Springer, Heidelberg (2003)
12. ElSalamouny, E., Krukow, K.T., Sassone, V.: An analysis of the exponential decay principle in probabilistic trust models. Theor. Comput. Sci. 410(41), 4067–4084 (2009)
13. ElSalamouny, E., Sassone, V., Nielsen, M.: HMM-based trust model. In: Degano, P., Guttman, J.D. (eds.) Formal Aspects in Security and Trust. LNCS, vol. 5983, pp. 21–35. Springer, Heidelberg (2010)
14. Evans, N.S., Dingledine, R., Grothoff, C.: A practical congestion attack on Tor using long paths. In: Proceedings of the 18th USENIX Security Symposium (2009)
15. Freedman, M.J., Morris, R.: Tarzan: a peer-to-peer anonymizing network layer. In: Atluri, V. (ed.) ACM Conference on Computer and Communications Security, pp. 193–206. ACM, New York (2002)
16. Halpern, J.Y., O'Neill, K.R.: Anonymity and information hiding in multiagent systems. Journal of Computer Security 13(3), 483–512 (2005)
17. Hamadou, S., Palamidessi, C., Sassone, V., ElSalamouny, E.: Probable innocence in the presence of independent knowledge. In: Degano, P., Guttman, J.D. (eds.) Formal Aspects in Security and Trust. LNCS, vol. 5983, pp. 141–156. Springer, Heidelberg (2010)
18. Hopper, N., Vasserman, E.Y., Chan-Tin, E.: How much anonymity does network latency leak? ACM Trans. Inf. Syst. Secur. 13(2) (2010)

19. Jakobsson, M.: Flash mixing. In: Annual ACM Symposium on Principles of Distributed Computing, PODC 1999, pp. 83–89 (1999)
20. Krukow, K., Nielsen, M., Sassone, V.: A logical framework for history-based access control and reputation systems. Journal of Computer Security 16(1), 63–101 (2008)
21. McLachlan, J., Hopper, N.: Don't clog the queue! circuit clogging and mitigation in P2P anonymity schemes. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 31–46. Springer, Heidelberg (2008)
22. McLachlan, J., Tran, A., Hopper, N., Kim, Y.: Scalable onion routing with Torsk. In: Al-Shaer, E., Jha, S., Keromytis, A.D. (eds.) ACM Conference on Computer and Communications Security, pp. 590–599. ACM, New York (2009)
23. Murdoch, S.J., Danezis, G.: Low-cost traffic analysis of tor. In: IEEE Symposium on Security and Privacy, pp. 183–195. IEEE Computer Society, Los Alamitos (2005)
24. Nambiar, A., Wright, M.: Salsa: a structured approach to large-scale anonymity. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) ACM Conference on Computer and Communications Security, pp. 17–26. ACM, New York (2006)
25. Neff, C.A.: A verifiable secret shuffle and its application to e-voting. In: ACM Conference on Computer and Communications Security, pp. 116–125 (2001)
26. Ohkubo, M., Abe, M.: A length-invariant hybrid mix. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 178–191. Springer, Heidelberg (2000)
27. Pappas, V., Athanasopoulos, E., Ioannidis, S., Markatos, E.P.: Compromising anonymity using packet spinning. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 161–174. Springer, Heidelberg (2008)
28. Reiter, M.K., Rubin, A.D.: Crowds: Anonymity for web transactions. ACM Trans. Inf. Syst. Secur. 1(1), 66–92 (1998)
29. Rennhard, M., Plattner, B.: Introducing MorphMix: peer-to-peer based anonymous internet usage with collusion detection. In: Jajodia, S., Samarati, P. (eds.) Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society, WPES, pp. 91–102. ACM, New York (2002)
30. Sassone, V., ElSalamouny, E., Hamadou, S.: Trust in Crowds: probabilistic behaviour in anonymity protocols. In: Symposium on Trustworthy Global Computing, TGC 2010. LNCS, vol. 6084. Springer, Heidelberg (2010)
31. Singh, A., Liu, L.: Trustme: Anonymous management of trust relationships in decentralized P2P systems. In: Shahmehri, N., Graham, R.L., Caronni, G. (eds.) Peer-to-Peer Computing, pp. 142–149. IEEE Computer Society, Los Alamitos (2003)
32. Syverson, P.F., Goldschlag, D.M., Reed, M.G.: Anonymous connections and onion routing. In: IEEE Symposium on Security and Privacy, pp. 44–54. IEEE Computer Society, Los Alamitos (1997)
33. Wang, Y., Vassileva, J.: Trust and reputation model in peer-to-peer networks. In: Shahmehri, N., Graham, R.L., Caronni, G. (eds.) Peer-to-Peer Computing. IEEE Computer Society, Los Alamitos (2003)
34. Wiangsripanawan, R., Susilo, W., Safavi-Naini, R.: Design principles for low latency anonymous network systems secure against timing attacks. In: Brankovic, L., Coddington, P.D., Roddick, J.F., Steketee, C., Warren, J.R., Wendelborn, A.L. (eds.) Proc. Fifth Australasian Information Security Workshop (Privacy Enhancing Technologies), AISW 2007. CRPIT, vol. 68, pp. 183–191. Australian Computer Society (2007)

Learning I/O Automata*

Fides Aarts and Frits Vaandrager

Institute for Computing and Information Sciences, Radboud University Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands

Abstract. Links are established between three widely used modeling frameworks for reactive systems: the ioco theory of Tretmans, the interface automata of De Alfaro and Henzinger, and Mealy machines. It is shown that, by exploiting these links, any tool for active learning of Mealy machines can be used for learning I/O automata that are deterministic and output determined. The main idea is to place a transducer in between the I/O automata teacher and the Mealy machine learner, which translates concepts from the world of I/O automata to the world of Mealy machines, and vice versa. The transducer comes equipped with an interface automaton that allows us to focus the learning process on those parts of the behavior that can effectively be tested and/or are of particular interest. The approach has been implemented on top of the LearnLib tool and has been applied successfully to three case studies.

1 Introduction

Model-based system development is becoming an increasingly important driving force in the software and hardware industry. In this approach, models become the primary artifacts throughout the engineering lifecycle of computer-based systems. Requirements, behavior, functionality, construction and testing strategies of computer-based systems are all described in terms of models. Models are not only used to reason about a system, but also used to allow all stakeholders to participate in the development process and to communicate with each other, to generate implementations, and to facilitate reuse. The construction of models typically requires significant manual effort, implying that in practice often models are not available, or become outdated as the system evolves. Automated support for constructing behavioral models of implemented components would therefore be extremely useful.

The problem of inducing, learning or inferring grammars and automata has been studied for decades, but only in recent years *grammatical inference* a.k.a. *grammar induction* has emerged as an independent field with connections to many scientific disciplines, including bio-informatics, computational linguistics and pattern recognition [10]. Also recently, some important developments have taken place on the borderline of verification, testing and machine learning, see

* This research was supported by European Community's Seventh Framework Programme under grant agreement no 214755 (QUASIMODO).

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 71–85, 2010.

e.g. [6,16,23], and researchers have shown that it is possible (at least in principle) to apply grammatical inference for learning models of software components. Grammatical inference techniques aim at building a grammar or automaton for an unknown language, given some data about this language. Within the setting of active learning, it is assumed that a *learner* interacts with a *teacher*. Inspired by work of Angluin [5] on the L^* algorithm, Niese [20] developed an adaptation of the L^* algorithm for active learning of deterministic Mealy machines. This algorithm has been further optimized in [23]. In the algorithm it is assumed that the teacher knows a deterministic Mealy machine $\mathcal{M}$. Initially, the learner only knows the action signature (the sets of input and output symbols I and O) and her task is to learn a Mealy machine that is equivalent to $\mathcal{M}$. The teacher will answer two types of questions — *output queries* ("what is the output generated in response to input $i \in I$?") and *equivalence queries* ("is an hypothesized machine $\mathcal{H}$ correct, i.e., equivalent to the machine $\mathcal{M}$?"). The learner always records the current state q of Mealy machine $\mathcal{M}$. In response to query i, the current state is updated to q' and answer o is returned to the learner. At any point the learner can "reset" the teacher, that is, change the current state back to the initial state of $\mathcal{M}$. The answer to an equivalence query $\mathcal{H}$ is either **yes** (in case $\mathcal{M} \approx \mathcal{H}$) or **no** (in case $\mathcal{M} \not\approx \mathcal{H}$). Furthermore, the teacher will give the learner a counterexample that proves that the learner's hypothesis is wrong with every negative equivalence query response, that is, an input sequence $u \in I^*$ such that $obs_{\mathcal{M}}(u) \neq obs_{\mathcal{H}}(u)$. This algorithm has been implemented in the LearnLib tool [23]. In practice, when a real implementation is used instead of an idealized teacher, the implementation cannot answer equivalence queries. Therefore, LearnLib "approximates" such queries by generating a long test sequence that is computed by standard methods such as state cover, transition cover, W-method, and the UIO method (see [15]). LearnLib has been applied successfully to learn computer telephony integrated (CTI) systems [11], and more recently to learn parts of the SIP and TCP protocols [1] and the new biometric passport [2].

Currently, LearnLib is able to automatically learn Mealy machines with up to 30.000 states. Nevertheless, a lot of further research will be required to make automata based learning tools suitable for routine use on industrial case studies. An important issue, clearly, is the development of abstraction techniques in order to be able to learn much larger state spaces (see [1], also for further references). Another issue is the extension of automata learning techniques to nondeterministic systems (see e.g. [29]). In this paper, we address a third issue that hinders the application of the LearnLib tool. In practice, the restriction of Mealy machines that each input corresponds to exactly one output is felt as being overly restrictive. Sometimes several inputs are required before a single output occurs, sometimes a single input triggers multiple outputs, etc.

The I/O automata of Lynch & Tuttle [18,17] and Jonsson [13] constitute a popular modelling framework which does not suffer from the restriction that inputs and outputs have to alternate. Our aim is to to develop efficient algorithms for active learning of I/O automata. Hence we assume that the teacher knows an I/O automaton $\mathcal{A}$. We consider a setting in which the task of the learner is to *partially* learn $\mathcal{A}$. More specifically, we assume that the learner initially knows an interface automaton $\mathcal{P}$ in the sense of De Alfaro and Henzinger [8], called the *learning purpose*, and that she has to learn the part of $\mathcal{A}$ whose behavior is compatible with $\mathcal{P}$. We think there are several good reasons to extend the framework of active learning with a notion of a learning purpose. In principle, the systems that we model using I/O automata will accept any input in any state. But in practice, a learner may not be able (e.g., not fast enough) to effectively provide any input in any state. Also, systems are often designed to be used in a certain manner, and their behavior may become unspecified and/or nondeterministic when they are used improperly. In such cases a learner may decide to interact with the system following the specified interaction protocol, for instance "after providing an input a user should wait for the system to become quiescent before she may provide a next input". A final motivation for using learning purposes is that often the state space of practical systems is very big and cannot be fully explored. By not providing certain inputs (in certain states), the learner may focus on interesting parts of the behavior that can be effectively learned.

Rather than developing and implementing an algorithm from scratch, we will use LearnLib. Our idea is to place a *transducer* in between the IOA teacher and the Mealy machine learner, which translates concepts from the world of I/O automata to the world of Mealy machines, and vice versa. The transducer and Mealy machine learner together then implement an IOA learner. Note that this architecture is very similar to the architecture proposed in [1], where a transducer is used to relate the large parameter spaces of realistic communication protocols to small sets of actions that can effectively be handled by state-of-the-art automata learning tools.

As a spin-off of our research, we establish links between three widely used modeling frameworks for reactive systems: the ioco theory of Tretmans [26,27], the interface automata of De Alfaro and Henzinger [8], and Mealy machines. In particular, we present behavior preserving maps between interface automata and Mealy machines, and we link the ioco preorder to alternating simulation.

The rest of this paper is structured as follows. Section 2 recalls interface automata and links alternating simulation to the ioco preorder. Section 3 addresses a basic question: what is the I/O behavior of an I/O automaton? Section 4 recalls Mealy machines and discusses translations between interface automata and Mealy machines. Section 5 describes our framework for learning I/O automata. In Section 6, we describe the implementation of our approach and its application to three case studies. Finally, Section 7 wraps up with some conclusions and suggestions for further research.

2 Interface Automata

An interface automaton models a reactive system that can interact with its environment. It is a simple type of state machine in which the transitions are associated with named actions. The actions are classified as either input or output.

The output actions are assumed to be under the control of the system whereas the input actions are under control of the environment. The interface automata that we study in this paper are a simplified version of the interface automata of De Alfaro and Henzinger [8] without internal actions. Within ioco theory [26,27] interface automata are called labelled transition systems with inputs and outputs. Interface automata are similar to the I/O automata of Lynch & Tuttle [18,17] and Jonsson [13]. The main difference is that in an I/O automaton input actions are required to be enabled in every state. In an interface automata certain input actions may be illegal in certain states: they are not enabled and we assume that the environment will not generate such inputs.

In this paper, an *interface automaton (IA)* is defined to be a tuple $\mathcal{A} = (I, O, Q, q^0, \rightarrow)$, where I, O and Q are finite, nonempty sets of input actions, output actions, and states, respectively, with I and O disjoint, $q^0 \in Q$ the initial state, and $\rightarrow \subseteq Q \times (I \cup O) \times Q$ the transition relation. We write $q \xrightarrow{a} q'$ if $(q, a, q') \in \rightarrow$. An action a is *enabled* in state q, notation $q \xrightarrow{a}$, if $q \xrightarrow{a} q'$, for some state q'. We write $out_{\mathcal{A}}(q)$, or just $out(q)$ if $\mathcal{A}$ is clear from the context, for the set $\{a \in O \mid q \xrightarrow{a}\}$ of output actions enabled in state q. For $S \subseteq Q$ a set of states, we write $out_{\mathcal{A}}(S)$ for $\bigcup\{out_{\mathcal{A}}(q) \mid q \in S\}$. An *I/O automaton (IOA)* is an interface automaton in which each input action is enabled in each state, that is $q \xrightarrow{i}$, for all $q \in Q$ and all $i \in I$. A state q is called *quiescent* if it enables no output action. An interface automaton $\mathcal{A}$ is

- *input deterministic* if for each state $q \in Q$ and for each action $a \in I$ there is at most one outgoing transition of q with label a: $q \xrightarrow{a} q_1 \wedge q \xrightarrow{a} q_2 \;\Rightarrow\; q_1 = q_2$;
- *output deterministic* if for each state $q \in Q$ and for each action $a \in O$ there is at most one outgoing transition of q with label a: $q \xrightarrow{a} q_1 \wedge q \xrightarrow{a} q_2 \;\Rightarrow\; q_1 = q_2$;
- *deterministic* if it is both input and output deterministic;
- *output determined* if each state has at most one outgoing output transition:

$$q \xrightarrow{o_1} q_1 \wedge q \xrightarrow{o_2} q_2 \wedge \{o_1, o_2\} \subseteq O \Rightarrow o_1 = o_2 \wedge q_1 = q_2.$$

Figure 1 displays a simple example of a deterministic IOA that is also output determined. The initial state is marked with an extra circle, there is a single input action *in* and there are two output actions *out*1 and *out*2.

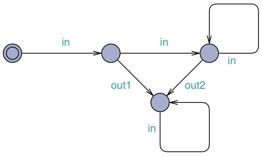


Fig. 1. A deterministic, output determined IOA

Let $\mathcal{A}_1 = (I, O, Q_1, q_1^0, \rightarrow_1)$ and $\mathcal{A}_2 = (I, O, Q_2, q_2^0, \rightarrow_2)$ be interface automata with the same signature. Let $A = I \cup O$ and let $X, Y \subseteq A$. A binary relation $R \subseteq Q_1 \times Q_2$ is an XY-*simulation* from $\mathcal{A}_1$ to $\mathcal{A}_2$ if whenever $(q, r) \in R$ and $a \in A$ it holds that:

- if $q \xrightarrow{a}_1 q'$ and $a \in X$ then there exists a $r' \in Q_2$ s.t. $r \xrightarrow{a}_2 r'$ and $(q', r') \in R$.
- if $r \xrightarrow{a}_2 r'$ and $a \in Y$ then there exists a $q' \in Q_1$ s.t. $q \xrightarrow{a}_1 q'$ and $(q', r') \in R$.

We write $\mathcal{A}_1 \leq_{XY} \mathcal{A}_2$ if there exists an XY-simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$ that contains (q_1^0, q_2^0). AA-simulations are commonly known as *bisimulations* and OI-simulations are known as *alternating simulations* [4]. De Alfaro and Henzinger [8] propose alternating simulations as the primary notion of refinement for IAs. In their approach, one IA refines another if it has weaker input assumptions and stronger output guarantees. We often write $\mathcal{A}_1 \leq_a \mathcal{A}_2$ instead of $\mathcal{A}_1 \leq_{OI} \mathcal{A}_2$ and $\mathcal{A}_1 \approx_b \mathcal{A}_2$ instead of $\mathcal{A}_1 \leq_{AA} \mathcal{A}_2$. There are several obvious inclusions between the different preorder, e.g. it follows that $\mathcal{A}_1 \leq_{AY} \mathcal{A}_2$ implies $\mathcal{A}_1 \leq_{XY} \mathcal{A}_2$.

Figure 2 shows an example of an alternating simulation between two IAs with inputs $\{in1, in2\}$ and outputs $\{out1, out2, d\}$.

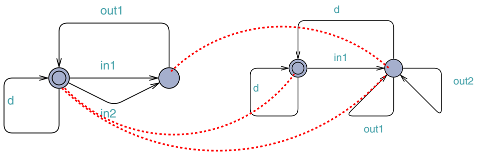


Fig. 2. Example of alternating simulation (from left to right IA)

Suppose that $\mathcal{A}_1 \leq_a \mathcal{A}_2$ and that R is the largest alternating simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$. We define $\mathsf{AS}(\mathcal{A}_1, \mathcal{A}_2)$, the *alternating simulation interface automaton* induced by $\mathcal{A}_1$ and $\mathcal{A}_2$, as the structure $(I, O, R, (q_1^0, q_2^0), \rightarrow)$ where

$$(q, r) \xrightarrow{a} (q', r') \Leftrightarrow q \xrightarrow{a}_1 q' \text{ and } r \xrightarrow{a}_2 r'.$$

Figure 3 shows the alternating simulation IA induced by the IAs of Figure 2. The following lemma follows easily from the definitions.

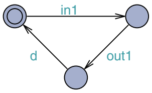


Fig. 3. IA induced by alternating simulation of Figure 2

Lemma 1. *Suppose* $\mathcal{A}_1 \leq_a \mathcal{A}_2$. *Then* $\mathcal{A}_1 \leq_{OA} \mathsf{AS}(\mathcal{A}_1, \mathcal{A}_2) \leq_{AI} \mathcal{A}_2$.

Larsen, Nyman and Wasowski [14] criticize interface automata and alternating simulations for being unable to express liveness properties and since they allow for trivial implementations: an IA $\mathcal{T}$ with a single state that accepts all inputs but never produces any output is a refinement of any IA over the same signature. In order to fix this problem, Larsen, Nyman and Wasowski [14] define model automata, an extension of interface automata with modality. In this paper, we propose a different solution, which is very simple and in the spirit of I/O automata and ioco theory: we make quiescence observable.

Let $\mathcal{A} = (I, O, Q, q^0, \rightarrow)$ be an IA and let δ be a fresh symbol (not in $I \cup O$). Then the *δ-extension* of $\mathcal{A}$, notation $\mathcal{A}^\delta$, is the IA obtained by adding δ-loops to all the quiescent states of $\mathcal{A}$. Formally, $\mathcal{A}^\delta = (I, O_\delta, Q, q^0, \rightarrow')$ where $O_\delta = O \cup \{\delta\}$ and $\rightarrow' = \rightarrow \cup \{q \xrightarrow{\delta} q \mid q \in Q \text{ quiescent}\}$. For $\mathcal{A}_1$ and $\mathcal{A}_2$ IAs with the same signature, we define $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2 \Leftrightarrow \mathcal{A}_1^\delta \leq_a \mathcal{A}_2^\delta$.

Observe that in general $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$ implies $\mathcal{A}_1 \leq_a \mathcal{A}_2$, but $\mathcal{A}_1 \leq_a \mathcal{A}_2$ does not imply $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$: even though $\mathcal{T} \leq_a \mathcal{A}$, for any IA $\mathcal{A}$ with the same signature as our trivial IA $\mathcal{T}$, we do in general not have $\mathcal{T} \leq_{a\delta} \mathcal{A}$. If $\mathcal{A}^\delta$ enables a sequence of input actions leading to a state r from which an output is possible, then $\mathcal{T}^\delta$ must enable the same sequence of inputs leading to a related state q. But whereas q enables a δ-transition, r does not enable a matching δ-transition. In order to argue that $\leq_{a\delta}$ indeed is a reasonable notion of implementation, we will now show that — under certain determinacy assumptions — $\leq_{a\delta}$ coincides with the well-known ioco preorder of [26,27].

We extend the transition relation to sequences by defining, for $\sigma \in (I \cup O)^*$, $\overset{\sigma}{\Rightarrow}$ to be the least relation that satisfies, for $q, q', q'' \in Q$ and $a \in I \cup O$,

$$q \overset{\epsilon}{\Rightarrow} q$$

$$q \overset{\sigma}{\Rightarrow} q' \wedge q' \xrightarrow{a} q'' \Rightarrow q \overset{\sigma\, a}{\Rightarrow} q''$$

Here ϵ denotes the empty sequence. We say that $\sigma \in (I \cup O)^*$ is a *trace* of $\mathcal{A}$ if $q^0 \overset{\sigma}{\Rightarrow} q$, for some state q, and write $Traces(\mathcal{A})$ for the set of traces of $\mathcal{A}$. We write $\mathcal{A}$ **after** σ for the set $\{q \in Q \mid q^0 \overset{\sigma}{\Rightarrow} q\}$ of states of $\mathcal{A}$ that can be reached with trace σ. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be IA with the same signature. Then $\mathcal{A}_1$ and $\mathcal{A}_2$ are *input-output conforming*, notation $\mathcal{A}_1$ **ioco** $\mathcal{A}_2$, if

$$\forall \sigma \in Traces(\mathcal{A}_2^\delta) : out(\mathcal{A}_1^\delta \textbf{ after } \sigma) \subseteq out(\mathcal{A}_2^\delta \textbf{ after } \sigma)$$

The results below link alternating simulation and the ioco preorder. These results generalize a similar, recent result of Veanes and Bjørner [28], which is stated in a setting of fully deterministic systems. We first state a small technical lemma.

Lemma 2. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be IAs with the same action signature such that $\mathcal{A}_1$ is input deterministic and $\mathcal{A}_2$ is output deterministic. Let R be an alternating simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$. Let $\sigma \in (I \cup O)^*$, $q_1 \in Q_1$ and $q_2 \in Q_2$ such that $q_1^0 \overset{\sigma}{\Rightarrow} q_1$ and $q_2^0 \overset{\sigma}{\Rightarrow} q_2$. Then $(q_1, q_2) \in R$.*

Theorem 1. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be IAs with the same action signature such that $\mathcal{A}_1$ is input deterministic and $\mathcal{A}_2$ is output deterministic. Then $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$ implies $\mathcal{A}_1$* **ioco** *$\mathcal{A}_2$.*

Theorem 2. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be IAs with the same action signature such that $\mathcal{A}_1$ is input enabled and $\mathcal{A}_2$ is deterministic. Then $\mathcal{A}_1$* **ioco** *$\mathcal{A}_2$ implies $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$.*

Corollary 1. *Let $\mathcal{A}_1$ be an input deterministic IOA and let $\mathcal{A}_2$ be a deterministic IA with the same action signature. Then $\mathcal{A}_1$* **ioco** *$\mathcal{A}_2$ iff $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$.*

Observe that all the determinacy conditions in the above results are essential: as soon as one assumption is left out the corresponding result no longer holds.

3 The I/O Behavior of I/O Automata

In order to be able to learn I/O automata, we need to decide which type of questions the learner may ask to the teacher. One obvious proposal would be to allow for membership queries of the form "Is sequence $u \in (I \cup O)^*$ a (quiescent) trace of the IOA?". However, there is strong evidence that this is an inefficient approach. In his PhD thesis [20], Niese compared two algorithms for learning Mealy machines. The first algorithm, an optimized version of Angluin's [5] L^* algorithm, allowed for membership queries "Is sequence $u \in (I \times O)^*$ a trace of the MM?". The second algorithm supported membership queries "What is the output generated by the MM in response to input sequence $u \in I^*$?". Niese showed that the second algorithm has a much better performance and requires less membership queries. We expect that for IOAs the situation is very similar.

Lynch & Tuttle [18,17] and Jonsson [13] do not define a notion of input/output behavior for I/O automata, that is, given a stream of input values that is provided by the environment, the stream of output values that is computed by the I/O automaton. The main reason for this is that such a notion of behavior is not compositional. Instead, the behavior of an IOA is defined in terms of traces, sequences of input and output actions that may be observed during runs of the automaton. Henzinger [9] links determinism to predictability and calls a reactive system *deterministic* if, for every stream of input values that is provided by the environment, the stream of output values that is computed by the system is unique. The example IOA of Figure 1 is not deterministic in this sense since the input stream *in in* may either lead to the output stream *out*1 or to the output stream *out*2. One obvious way to proceed is to restrict the class of IOA that one calls deterministic, and and to study a notion of input/output behavior for this restricted class. This route is explored by Panangaden and Stark [21] in their study of "monotone" processes. We will explore a different route, in which the power of testers is slightly increased and the IOA of Figure 1 becomes again behavior deterministic.

If a system is predictable then one may expect that, for any history of input and output actions, the time required by the system to produce its next output (if any) is more or less known. Predictability is at the basis of the assumption in ioco theory that quiescence is observable: whenever a test is carried out, it is assumed that if a system does not produce an output within some fixed time T after the last input, it will never produce an output. By the same line of reasoning, one could assume that there exists a fixed time t such that the system never produces an output within time t after an input. Hence, if one assumes that the tester can compute faster than the IUT, then in principle the tester always has the choice to either wait for the next output of the IUT or to generate its next input before time t, that is, before occurrence of the output. Based on these considerations, we slightly increase the power of the testers: at any point we let the tester decide who is going to perform the next step, the IUT or the tester itself.

Formally, we introduce a fresh *delay* action Δ. By performing Δ, the environment gives an IOA the opportunity to perform its next output (if any). Let

$I_\Delta = I \cup \{\Delta\}$. The behavior of an environment can then be described by an *environment sequence* in $(I_\Delta)^*$, that is, a sequence of input actions interleaved with delay actions. Let $\mathcal{A} = (I, O, Q, q^0, \rightarrow)$ be an IA and let $q, q' \in Q$, $e \in (I_\Delta)^*$ and $u \in (I \cup O_\delta)^*$. We write $q \overset{e/u}{\Rightarrow} q'$ to indicate that as a result of offering environment sequence e in state q, $\mathcal{A}_\delta$ may produce trace u and end up in state q'. Formally, $\overset{e/u}{\Rightarrow}$ is the least relation that satisfies $q \overset{\epsilon/\epsilon}{\Rightarrow} q$ and:

$$q \overset{e/u}{\Rightarrow} q' \wedge q' \xrightarrow{i} q'' \wedge i \in I \Rightarrow q \overset{e\,i/u\,i}{\Rightarrow} q''$$
$$q \overset{e/u}{\Rightarrow} q' \wedge q' \xrightarrow{o} q'' \wedge o \in O_\delta \Rightarrow q \overset{e\,\Delta/u\,o}{\Rightarrow} q''$$

For each environment sequence $e \in (I_\Delta)^*$, we define $obs_{\mathcal{A}}(e)$ to be the set of traces that may be observed when offering e to $\mathcal{A}_\delta$, that is, $obs_{\mathcal{A}}(e) = \{u \in (I \cup O_\delta)^* \mid \exists q \in Q : q^0 \overset{e/u}{\Rightarrow} q\}$. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two IOAs with the same sets I and O of input and output actions, respectively, We write $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$, if $obs_{\mathcal{A}_1}(e) \subseteq obs_{\mathcal{A}_2}(e)$, for all environment sequences $e \in (I_\Delta)^*$. If $\mathcal{A}$ is a deterministic and output determined IOA then $obs_{\mathcal{A}}(e)$ contains exactly one element for each input sequence e. Thus, with this notion of observable behavior, a deterministic and output determined IOA is also behavior deterministic in the sense of Henzinger [9].

Even though our notion of observation is based on a stronger notion of testing than ioco theory, the resulting notion of preorder is the same.

Theorem 3. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be IOAs with the same inputs and outputs. Then $\mathcal{A}_1$* **ioco** *$\mathcal{A}_2$ iff $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$.*

4 From Interface Automata to Mealy Machines and Back

A *(nondeterministic) Mealy machine (MM)* is a tuple $\mathcal{M} = (I, O, Q, q^0, \rightarrow)$, where I, O and Q are finite, nonempty sets of inputs, outputs, and states, respectively, $q^0 \in Q$ is the initial state, and $\rightarrow \subseteq Q \times I \times O \times Q$ is the transition relation. We write $q \xrightarrow{i/o} q'$ if $(q, i, o, q') \in \rightarrow$, and $q \xrightarrow{i/o}$ if there exists a q' such that $q \xrightarrow{i/o} q'$. Mealy machines are assumed to be *input enabled*: for each state q and input i, there exists an output o such that $q \xrightarrow{i/o}$. The transition relation is extended to sequences by defining $\overset{u/s}{\Rightarrow}$ to be the least relation that satisfies, for $q, q', q'' \in Q$, $u \in I^*$, $s \in O^*$, $i \in I$, and $o \in O$: $q \overset{\epsilon/\epsilon}{\Rightarrow} q$ and $q \overset{u/s}{\Rightarrow} q' \wedge q' \xrightarrow{i/o} q'' \quad \Rightarrow \quad q \overset{u\,i/s\,o}{\Rightarrow} q''$. A state $q \in Q$ is called *reachable* if $q^0 \overset{u/s}{\Rightarrow} q$, for some u and s. A Mealy machine is *deterministic* iff given a state q and an input i there is exactly one output o and exactly one state q' such that $q \xrightarrow{i/o} q'$.

For $q \in Q$ and $u \in I^*$, define $obs_{\mathcal{M}}(q, u)$ to be the set of output sequences that may be produced when offering input sequence u to $\mathcal{M}$, that is, $obs_{\mathcal{M}}(q, u) = \{s \in O^* \mid \exists \overline{q} : q \overset{u/s}{\Rightarrow} \overline{q}\}$. Two states $q, q' \in Q$ are *observation equivalent*, notation

$q \approx q'$, if $obs_{\mathcal{M}}(q,u) = obs_{\mathcal{M}}(q',u)$, for all input strings $u \in I^*$. Write $obs_{\mathcal{M}}(u)$ as a shorthand for $obs_{\mathcal{M}}(q^0,u)$. Two Mealy machines $\mathcal{M}_1$ and $\mathcal{M}_2$ with the same sets of inputs I are *observation equivalent*, notation $\mathcal{M}_1 \approx \mathcal{M}_2$, if $obs_{\mathcal{M}_1}(u) = obs_{\mathcal{M}_2}(u)$, for all input strings $u \in I^*$. If $\mathcal{M}$ is deterministic then $obs_{\mathcal{M}}(u)$ is a singleton set for each input sequence u. Thus a deterministic Mealy machine is also behavior deterministic in the sense of Henzinger [9].

We call an interface automaton *active* if each state enables an output action. Observe that for any interface automaton $\mathcal{A}$, the δ-extension $\mathcal{A}^\delta$ is active. Active interface automata can be translated to equivalent Mealy machines. We translate each input transition $q \xrightarrow{i} q'$ to a transition $q \xrightarrow{i/+} q'$ in the Mealy machine, where $+$ is a fresh output action denoting that the input is accepted. If input i is not enabled in state q, then we add a self-loop $q \xrightarrow{i/-} q$ to the Mealy machine. Here $-$ is a fresh output symbol denoting that the input is illegal. The fresh input action Δ ("delay") is used to probe for possible outputs: each output transition $q \xrightarrow{o} q'$ translates to a transition $q \xrightarrow{\Delta/o} q'$ in the Mealy machine.

Formally, for active IA $\mathcal{A} = (I, O, Q, q^0, \rightarrow)$, the Mealy machine $T(\mathcal{A})$ is defined as the structure $(I_\Delta, O \cup \{+,-\}, Q, q^0, \rightarrow')$, where

$$\begin{aligned} i \in I \wedge q \xrightarrow{i} q' &\Rightarrow q \overset{i/+}{\rightarrow'} q' \\ i \in I \wedge q \overset{i}{\not\rightarrow} &\Rightarrow q \overset{i/-}{\rightarrow'} q \\ o \in O \wedge q \xrightarrow{o} q' &\Rightarrow q \overset{\Delta/o}{\rightarrow'} q' \end{aligned}$$

Figure 4 illustrates transformation T. We now define transformation R, the inverse of transformation T, which takes a Mealy machine and turns it into an IA. Let $\mathcal{M} = (I_\Delta, O \cup \{+,-\}, Q, q^0, \rightarrow)$ be a Mealy machine. Then $R(\mathcal{M})$ is the IA $(I, O, Q, q^0, \rightarrow')$, where

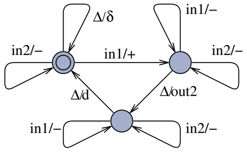


Fig. 4. Result of applying T to the δ-extension of the IA of Figure 3

$$i \in I \wedge q \xrightarrow{i/+} q' \;\Rightarrow\; q \overset{i}{\rightarrow'} q' \quad \text{and} \quad o \in O \wedge q \xrightarrow{\Delta/o} q' \;\Rightarrow\; q \overset{o}{\rightarrow'} q'$$

If one takes any total IA $\mathcal{A}$ and applies first T and then R, one gets back $\mathcal{A}$.

Theorem 4. *Let $\mathcal{A}$ be a total IA. Then $\mathcal{A} = R(T(\mathcal{A}))$.*

Observe that if $\mathcal{A}$ is deterministic and output determined then $T(\mathcal{A})$ is deterministic, and if $\mathcal{M}$ is deterministic then $R(\mathcal{M})$ is deterministic and output determined. In order to obtain a dual version of Theorem 4, we need to impose three additional conditions on $\mathcal{M}$. Let $\mathcal{M}$ be a Mealy machine whose inputs include Δ and whose outputs include $+$ and $-$. Then $\mathcal{M}$ is *separated* if an input in I always leads to an output $+$ or $-$, and input Δ always leads to an output in O: $q \xrightarrow{i/o} q' \;\Rightarrow\; (i = \Delta \Leftrightarrow o \in O)$. $\mathcal{M}$ is *consistent* if there exists no state q and input i for which both outputs $+$ and $-$ are possible: $\neg(q \xrightarrow{i/+} \wedge q \xrightarrow{i/-})$. $\mathcal{M}$ is *stable* if an output $-$ does not lead to a change of state: $q \xrightarrow{i/-} q' \;\Rightarrow\; q = q'$. Clearly, for any total IA $\mathcal{A}$, $T(\mathcal{A})$ is separated, consistent and stable. Note that deterministic Mealy machines are consistent. Using the conditions of separation, consistency and stability, it is easy to prove $\mathcal{M} = T(R(\mathcal{M}))$.

Theorem 5. *Let $\mathcal{M}$ be a separated, consistent and stable Mealy machine with inputs I_Δ and outputs $O \cup \{+,-\}$. Then $\mathcal{M} = T(R(\mathcal{M}))$.*

5 Learning I/O Automata

In this section, we present our approach for active learning of I/O automata. We assume that the teacher knows a deterministic and output determined IOA $\mathcal{A} = (I, O, Q, q^0, \rightarrow)$. We consider a setting in which the task of the learner is to *partially* learn $\mathcal{A}$: the learner initially knows a deterministic interface automaton $\mathcal{P} = (I, O_\delta, P, p^0, \rightarrow')$, called the *learning purpose*, and has to learn the part of $\mathcal{A}$ whose behavior is compatible with $\mathcal{P}$. We require $\mathcal{A}^\delta \leq_a \mathcal{P}$.

The teacher and learner play the following game. The teacher records the current state of $\mathcal{A}$, which initially is q^0, and the learner records the current state of $\mathcal{P}$, which initially is p^0. Suppose that the teacher is in state q and the learner is in state p. The learner now can do four things: (1) If an input transition $p \overset{i}{\rightarrow}{}' p'$ is enabled then it may jump to p' and present input i to the teacher, which will then jump to the state q' such that $q \xrightarrow{i} q'$. (2) The learner may present a delay Δ to the teacher. If the teacher enables some output o, then it will jump to the unique state q' such that $q \xrightarrow{o} q'$ and return answer o to the learner. If no output action is enabled in q then the teacher returns δ. The learner then jumps to the unique state p' that can be reached by the answer o or δ that it has just received (by the assumption that $\mathcal{A}^\delta \leq_a \mathcal{P}$ we know this state exists). (3) The learner may return to its initial state and ask the teacher to do the same ("‘reset”). (4) The learner may pose a preorder query (“is an hypothesized IA $\mathcal{H}$ correct?”). An hypothesis is a deterministic, output determined IA $\mathcal{H}$ such that $\mathcal{H}^\delta \leq_{AI} \mathcal{P}$. An hypothesis is correct if $\mathcal{A} \leq_{a\delta} \mathcal{H}$. If $\mathcal{H}$ is correct then the teacher returns the answer **yes**. If an hypothesis is not correct then, by Corollary 1, $\mathcal{H}^\delta$ has a trace σ such that the unique output o enabled by $\mathcal{A}^\delta$ after σ differs from the unique output enabled by $\mathcal{H}^\delta$ after σ. The teacher then returns the answer **no** together with counterexample $\sigma\ o$.

In order to appreciate our learning framework, consider the trivial learning purpose $\mathcal{P}_{triv}$ displayed in Figure 5 (left). Here notation $i : I$ means that we have an instance of the transition for each input $i \in I$. Notation $o : O$ is defined

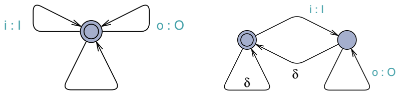


Fig. 5. A trivial learning purpose (left) and a learning purpose with a nontrivial δ transition (right)

similarly. If $\mathcal{H}$ is an hypothesis, then by definition $\mathcal{H}^\delta \leq_{AI} \mathcal{P}_{triv}$. This just means that $\mathcal{H}$ is input enabled. If $\mathcal{H}$ is correct then $\mathcal{A} \leq_{a\delta} \mathcal{H}$. Since both $\mathcal{A}$ and $\mathcal{H}$ are deterministic, output determined IAs, this means that $\mathcal{A}$ and $\mathcal{H}$ are bisimilar! The following lemma provides some key insight in our approach in case of an arbitrary learning purpose. It implies that if hypothesis $\mathcal{H}$ is correct, $\mathcal{H}^\delta$ is bisimilar to $\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P})$.

Lemma 3. *Suppose $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ are IAs, $\mathcal{A}_1$ is active and input deterministic, $\mathcal{A}_2$ is output determined, $\mathcal{A}_3$ is output deterministic, and $\mathcal{A}_1 \leq_a \mathcal{A}_3 \leq_{AI} \mathcal{A}_2$. Then $\mathcal{A}_3 \approx_b \mathsf{AS}(\mathcal{A}_1, \mathcal{A}_2)$.*

It is important that a learning purpose may contain nontrivial δ transitions. As an example, consider the IA of Figure 5 (right). This learning purpose expressing that after an input one has to wait until the system gets into a quiescent state before offering the next input. It is not possible to express this without δ's. But since in the end we want to learn IAs without δ's, we need an operation that eliminates all δ-transitions from an automaton. Let $\mathcal{A} = (I, O_\delta, Q, q^0, \rightarrow)$ be an IA. Let $\equiv$ be the smallest equivalence relation that contains $\xrightarrow{\delta}$. Then we define $\rho(\mathcal{A})$ to be the quotient IA $(I, O, Q/\!\equiv, q^0/\equiv, \rightarrow')$ where

$$q/\equiv \xrightarrow{a} q'/\equiv \Leftrightarrow \exists r, r' : q \equiv r \wedge r \xrightarrow{a} r' \wedge r' \equiv q'$$

The following lemma implies that under certain conditions operation ρ preserves bisimulation equivalence.

Lemma 4. *Suppose $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic, output determined IAs, $\mathcal{A}_1$ has outputs O, $\mathcal{A}_2$ has outputs O_δ, and both IAs share the same sets of inputs I. Suppose furthermore that $\mathcal{A}_2$ satisfies the following* triangle property*, for $i \in I$: $q \xrightarrow{\delta} q' \wedge q \xrightarrow{i} q'' \Rightarrow q' \xrightarrow{i} q''$. Then $\mathcal{A}_1^\delta \approx_b \mathcal{A}_2$ implies $\mathcal{A}_1 \approx_b \rho(\mathcal{A}_2)$.*

We always assume that the learning purpose $\mathcal{P}$ satisfies the triangle property. Under this assumption, it follows using the above lemma that, if hypothesis $\mathcal{H}$ is correct, $\mathcal{H}$ is bisimilar to $\rho(\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P}))$.

Rather than developing and implementing an algorithm from scratch, we use the LearnLib tool [23] to implement our learning approach. We place a *transducer* in between the IOA teacher and the Mealy machine learner, which records the current state p of the learning purpose $\mathcal{P}$ and translates concepts from the world of I/O automata to the world of Mealy machines, and vice versa, using the translation functions defined in the previous section. Initially, the MM learner only knows a signature consisting of inputs I_Δ and outputs $O_\delta \cup \{+,-\}$. The behavior of the transducer is as follows:

- Whenever the transducer receives an output query $i \in I$ from the MM learner, it checks if i is enabled in the current state of $\mathcal{P}$. If the input is not enabled ("illegal") then the transducer returns an output $-$ to the MM learner. If the output is enabled then the transducer returns an output $+$ to the MM learner, updates state p to the unique state p' with an i-transition from p to p', and forwards i to the IOA teacher.
- Whenever the transducer receives an output query Δ this is forwarded directly to the IOA teacher. When it receives a response $o \in O_\delta$, the transducer updates state p accordingly, and forwards o to the MM learner.
- Whenever the transducer receives a "reset" from the MM learner, it resets its state to p^0, and forwards the "reset" to the IOA teacher.
- Whenever the transducer receives an equivalence query $\mathcal{H}$ from the MM learner, then it first checks whether $\rho(R(\mathcal{H})) \leq_{AI} \mathcal{P}$ (since both IAs are deterministic, this can be done in time linear in the size of their synchronous product). If $\rho(R(\mathcal{H}))$ does not conform to learning purpose $\mathcal{P}$, then an answer **no** is returned to the MM learner, together with a distinguishing trace in which all output symbols are replaced by Δ. If $\rho(R(\mathcal{H})) \leq_{AI} \mathcal{P}$ then the transducer forwards the preorder query $\rho(R(\mathcal{H}))$ to the IOA teacher. The transducer forwards a subsequent response of the IOA teacher to the MM learner, but with all output symbols replaced by Δ. If the response is **yes** then the transducer has successfully learned an IA $\rho(R(\mathcal{H}))$ that meets all the requirements.

Observe that when LearnLib is used, equivalence queries are always separated and stable. We claim that the algorithm always terminates and that the transducer indeed learns an IOA that is equivalent to $\rho(\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P}))$. In order to see why this claim is true, a key observation is that the IOA teacher and transducer together behave like a teacher for Mealy machine $T(\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P}))$.

Lemma 5. *The IOA teacher and transducer together behave like a teacher for Mealy machine $T(\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P}))$.*

The main technical result of this article is that the MM learner and the transducer together will succeed in learning $\rho(\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P}))$, that is, the subautomaton of $\mathcal{A}$ induced by the learning purpose $\mathcal{P}$:

Theorem 6. *The composition of MM learner and transducer behaves like a learner for I/O automata, that is, execution will terminate after a finite number of queries, and upon termination the transducer has learned an IA that is bisimilar to $\rho(\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P}))$.*

6 Experiments

We have implemented and applied our approach to infer three types of I/O automata. In this section, we first describe our experimental setup, thereafter its application to the three case studies.[1]

To serve as IOA teacher, we read in an I/O automaton specified in Aldebaran format [3]. We connect the IOA teacher to a transducer equipped with an interface automaton, which is also described in Aldebaran format. As MM learner in our framework, we use the LearnLib library [22].

In our first (trivial) case study we learned the IOA shown in Figure 1. Because the interaction with this automaton is not constrained, we used an interface automaton that accepts every input and output, see Figure 5. The inferred Mealy machine model can be transformed to the IOA by transformations R and ρ.

A model of the electronic passport [12,7] has been inferred in a second experiment. We provided the IOA teacher with a model of the protocol taken from Mostowski et al. [19]. Analyzing the behavior of the automaton revealed that almost always the passport reacts like a Mealy machine: 13 out of 14 inputs generate an output symbol before a new input symbol can be transferred. Following this information, we defined an interface automaton in which inputs alternate with outputs or quiescence, see Figure 6 (left). Because no output is generated in response to a *Reset* input in the IOA, an output δ occurs within the Mealy machine that is learned. In fact, the inferred Mealy machine has one additional state, which can only be reached by a Δ/δ transition. After applying transformation R and ρ, we obtained the corresponding subautomaton of the IOA that was given to the teacher. With respect to learning performance, we observe that inferring an IOA requires more membership queries than learning the same behavior as a Mealy machine having i/o instead of $i/+$ and Δ/o transitions. Inferring an IOA of the electronic passport required 44294 membership queries, whereas learning the corresponding Mealy machine with i/o transitions merely needed 1079 queries. The difference can be explained by the fact that 80,72% of the membership queries asked to infer the passport IOA comprised unspecified input sequences. Because of the Mealy machine behavior of the IOA, no outputs are defined for most consecutive inputs. Moreover, membership queries were asked for the additional state.

In a third case study we applied our approach to learn a model of the Session Initiation Protocol (SIP) [25,24]. The teacher is supplied with an IOA based on a Mealy machine generated using inference and abstraction techniques [1]. Analyzing the structure of the automaton showed that each input symbol is followed by one or more outputs. Furthermore, in the initial state only certain inputs are allowed. To concentrate the learning on this restricted behavior, we used the interface automaton shown in Figure 6 (right). Again, by applying transformation ρ and R, the inferred Mealy machine could be converted to the corresponding subautomaton of the IOA given to the teacher.

[1] All IOAs and interface automata used in the different case studies as well as the corresponding learned Mealy machines can be found at the URL `http://www.mbsd.cs.ru.nl/publications/papers/fvaan/LearningIOAs/`.

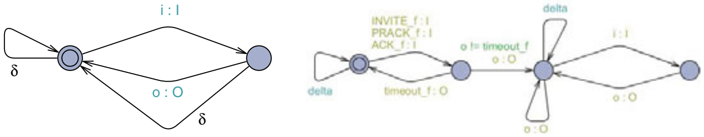


Fig. 6. IA in which each input is followed by at most one output (left) and IA in which initially only certain inputs are allowed and two consecutive inputs are not allowed (right)

7 Conclusions and Future Work

We have presented an approach for active learning of deterministic and output determined I/O automata. By eliminating the restriction from Mealy machines that inputs and outputs have to alternate, we have extended the class of models that can be learned. Our approach has been implemented on top of the LearnLib tool and has been applied successfully to three case studies. A new idea introduced in this paper is to use interface automata to focus the learning process to interesting/relevant parts of the behavior. Both in the passport and the SIP case study, the use of interface automata greatly reduced the number of queries. The efficiency of our learning approach can be improved by integrating this notion of interface automata within LearnLib: in this way it will be possible to further reduce the number of membership queries. Obvious topics for future research are to extend our approach to automata with nondeterminism and silent transitions, and to integrate our transducers with the ones used in [1] for data abstraction.

Acknowledgement. Many thanks to Bengt Jonsson, Bernhard Steffen, Jan Tretmans and the anonymous referees for inspiring discussions and/or pointers to the literature, and to Falk Howar for his generous LearnLib support.

References

1. Aarts, F.: Inference and Abstraction of Communication Protocols. Master thesis, Radboud University Nijmegen and Uppsala University (November 2009)
2. Aarts, F., Schmaltz, J., Vaandrager, F.: Inference and abstraction of the biometric passport (May 2010)
3. ALDEBARAN manual, `http://www.inrialpes.fr/vasy/cadp/man/aldebaran.html`
4. 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)
5. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. 75(2), 87–106 (1987)
6. Berg, T., Grinchtein, O., Jonsson, B., Leucker, M., Raffelt, H., Steffen, B.: On the correspondence between conformance testing and regular inference. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 175–189. Springer, Heidelberg (2005)
7. BSI. Advanced security mechanisms for machine readable travel documents - extended access control (eac) - version 1.11. Technical Report TR-03110, German Federal Office for Information Security (BSI), Bonn, Germany (2008)
8. de Alfaro, L., Henzinger, T.: Interface automata. In: ESEC/FSE 2001. Software Engineering Notes, vol. 26, pp. 109–120. ACM Press, New York (2001)

9. Henzinger, T.: Two challenges in embedded systems design: Predictability and robustness. Philosophical Trans. of the Royal Society A 366, 3727–3736 (2008)
10. Higuera, C.d.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, Cambridge (2010)
11. Hungar, H., Niese, O., Steffen, B.: Domain-specific optimization in automata learning. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 315–327. Springer, Heidelberg (2003)
12. ICAO. Doc 9303 - machine readable travel documents - part 1-2. Technical report, International Civil Aviation Organization, 6th edn. (2006)
13. Jonsson, B.: Modular verification of asynchronous networks. In: PODC 1987, pp. 152–166 (1987)
14. Larsen, K., Nyman, U., Wasowski, A.: Modal i/o automata for interface and product line theories. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 64–79. Springer, Heidelberg (2007)
15. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines — a survey. Proceedings of the IEEE 84(8), 1090–1123 (1996)
16. Leucker, M.: Learning meets verification. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2006. LNCS, vol. 4709, pp. 127–151. Springer, Heidelberg (2007)
17. Lynch, N.: Distributed Algorithms. Morgan Kaufmann Publishers, Inc., San Francisco (1996)
18. Lynch, N., Tuttle, M.: Hierarchical correctness proofs for distributed algorithms. In: PODC 1987, pp. 137–151 (1987)
19. Mostowski, W., Poll, E., Schmaltz, J., Tretmans, J., Wichers Schreur, R.: Model-based testing of electronic passports. In: FMICS 2009, pp. 207–209. Springer, Heidelberg (2009)
20. Niese, O.: An Integrated Approach to Testing Complex Systems. PhD thesis, University of Dortmund (2003)
21. Panangaden, P., Stark, E.: Computations, residuals, and the power of indeterminancy. In: Lepistö, T., Salomaa, A. (eds.) ICALP 1988. LNCS, vol. 317, pp. 439–454. Springer, Heidelberg (1988)
22. Raffelt, H., Steffen, B., Berg, T.: Learnlib: a library for automata learning and experimentation. In: FMICS 2005, pp. 62–71. ACM Press, New York (2005)
23. Raffelt, H., Steffen, B., Berg, T., Margaria, T.: Learnlib: a framework for extrapolating behavioral models. STTT 11(5), 393–407 (2009)
24. Rosenberg, J., Schulzrinne, H.: Reliability of Provisional Responses in Session Initiation Protocol (SIP). RFC 3262 (Proposed Standard) (June 2002)
25. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard). Updated by RFCs 3265, 3853, 4320, 4916, 5393 (June 2002)
26. Tretmans, J.: Test generation with inputs, outputs, and repetitive quiescence. Software–Concepts and Tools 17, 103–120 (1996)
27. Tretmans, J.: Model based testing with labelled transition systems. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) FORTEST. LNCS, vol. 4949, pp. 1–38. Springer, Heidelberg (2008)
28. Veanes, M., Bjørner, N.: Input-output model programs. In: Leucker, M., Morgan, C. (eds.) Theoretical Aspects of Computing - ICTAC 2009. LNCS, vol. 5684, pp. 322–335. Springer, Heidelberg (2009)
29. Willemse, T.: Heuristics for ioco-based test-based modelling. In: Brim, L., Haverkort, B.R., Leucker, M., van de Pol, J. (eds.) FMICS 2006 and PDMC 2006. LNCS, vol. 4346, pp. 132–147. Springer, Heidelberg (2007)

Constrained Monotonic Abstraction: A CEGAR for Parameterized Verification

Parosh Aziz Abdulla[1], Yu-Fang Chen[2], Giorgio Delzanno[3], Frédéric Haziza[1], Chih-Duo Hong[2], and Ahmed Rezine[1]

[1] Uppsala University, Sweden
[2] Academia Sinica, Taiwan
[3] Università di Genova, Italy

Abstract. In this paper, we develop a counterexample-guided abstraction refinement (CEGAR) framework for *monotonic abstraction*, an approach that is particularly useful in automatic verification of safety properties for *parameterized systems*. The main drawback of verification using monotonic abstraction is that it sometimes generates spurious counterexamples. Our CEGAR algorithm automatically extracts from each spurious counterexample a set of configurations called a "Safety Zone"and uses it to refine the abstract transition system of the next iteration. We have developed a prototype based on this idea; and our experimentation shows that the approach allows to verify many of the examples that cannot be handled by the original monotonic abstraction approach.

1 Introduction

We investigate the analysis of safety properties for *parameterized systems*. A parameterized system consists of an arbitrary number of identical finite-state processes running in parallel. The task is to verify correctness regardless of the number of processes.

One of the most widely used frameworks for infinite-state verification uses *systems that are monotonic w.r.t. a well-quasi ordering* $\preceq$ [2,22]. This framework provides a scheme for proving termination of backward reachability analyses, which has already been used for the design of verification algorithms of various infinite-state systems (e.g., Petri nets, lossy channel systems) [8,20,21]. The main idea is the following. For a class of models, we find a preorder $\preceq$ on the set of configurations that satisfies the following two conditions (1) the system is monotonic w.r.t. $\preceq$ and (2) $\preceq$ is a well-quasi ordering (WQO for short). Then, backward reachability analysis from an upward closed set (w.r.t. $\preceq$) is guaranteed to terminate, which implies that the reachability problem of an upward closed set (w.r.t. $\preceq$) is decidable.

However, there are several classes of systems that do not fit into this framework, since it is hard to find a preorder that meets the aforementioned two conditions at the same time. An alternative solution is to first find a WQO $\preceq$ on the set of configurations and then apply *monotonic abstraction* [6,4,7] in order to *force* monotonicity. Given a preorder $\preceq$ on configurations, monotonic abstraction

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 86–101, 2010.

defines an abstract transition system for the considered model that is monotonic w.r.t. $\preceq$. More precisely, it considers a transition from a configuration c_1 to a configuration c_2 to be possible if there exists some smaller configuration $c_1' \preceq c_1$ that has a transition to c_2. The resulting abstract transition system is clearly monotonic w.r.t $\preceq$ and is an over-approximation of the considered model. Moreover, as mentioned, if $\preceq$ is a WQO, the termination of backward reachability analysis is guaranteed in the abstract transition system.

Monotonic abstraction has shown to be useful in the verification of heap manipulating programs [1] and parameterized systems such as *mutual exclusion* and *cache coherence* protocols [4,6]. In most of the benchmark examples for these classes, monotonic abstraction can generate abstract transition systems that are safe w.r.t to the desired properties (e.g. mutual exclusion). The reason is that, for these cases, we need only to keep track of simple constraints on individual variables in order to successfully carry out verification. However, there are several classes of protocols where we need more complicated invariants in order to avoid generating spurious counterexamples. Examples include cases where processes synchronize via *shared counters* (e.g. readers and writers protocol) or *reference counting* schemes used to handle a common set of resources (e.g. virtual memory management). For these cases, monotonic abstraction often produces spurious counterexamples, since it is not sufficiently precise to preserve the needed invariants. Therefore, we introduce in this paper a *counterexample-guided abstraction refinement (CEGAR)* approach to *automatically* and *iteratively* refine the abstract transition system and remove spurious counterexamples.

The idea of the CEGAR algorithm is as follows. It begins with an initial preorder $\preceq_0$, which is the one used in previous works on monotonic abstraction [6]. In the i-th iteration, it tries to verify the given model using monotonic abstraction w.r.t. the preorder $\preceq_{i-1}$. Once a *counterexample* is found in the abstract transition system, the algorithm simulates it on the concrete transition system. In case the counterexample is spurious, the algorithm extracts from it a set S of configurations called a "Safety Zone". The computation of the "Safety Zone"is done using *interpolation* [28,26]. The set S ("Safety Zone") is then used to *strengthen* the preorder that will be used in the next iteration. Monotonic abstraction produces a more accurate abstract transition system with the strengthened preorder. More precisely, in the $(i+1)$-th iteration, the algorithm works on an abstract transition system induced by monotonic abstraction and a preorder $\preceq_i := \{(c, c') | \; c \preceq_{i-1} c' \text{ and } c' \in S \Rightarrow c \in S\}$. Intuitively, the strengthened preorder forbids configurations inside a "Safety Zone"to use a transition from some smaller configuration (w.r.t $\preceq_{i-1}$) outside the "Safety Zone".

The strengthening of the preorder has an important property: It preserves WQO. That is, if $\preceq_{i-1}$ is a WQO, then $\preceq_i$ is also a WQO, for all $i > 0$. Therefore, the framework of monotonic systems w.r.t. a WQO can be applied to each abstract transition system produced by monotonic abstraction and hence termination is guaranteed for each iteration. Based on the method, we have implemented a prototype, and successfully used it to automatically verify several non-trivial examples, such as protocols synchronizing by shared counters and reference counting schemes, that cannot be handled by the original monotonic abstraction approach.

Outline. We define parameterized systems and their semantics in Section 2. In Section 3, we first introduce monotonic abstraction and then give an overview of the CEGAR algorithm. In Section 4, we describe the details of the CEGAR algorithm. We introduce a *symbolic representation* of infinite sets of configurations called *constraint*. In Section 4, we show that all the *constraint* operations used in our algorithm are computable. In Section 6, we show that the termination of backward reachability checking is guaranteed in our CEGAR algorithm. Section 7 describes some extension of our model for parameterized system. In Section 8 we describe our experimentation. Finally, in Section 9, we conclude with a discussion of related tools and future works.

2 Preliminaries

In this section, we define a model for parameterized systems. We use $\mathbb{B}$ to denote the set $\{true, false\}$ of Boolean values, $\mathbb{N}$ to denote the set of natural numbers, and $\mathbb{Z}$ to denote the set of integers. Let P be a set and $\preceq$ be a binary relation on P. The relation $\preceq$ is a *preorder* on P if it is reflexive and transitive. Let $Q \subseteq P$, we define a *strengthening* of $\preceq$ by Q, written $\preceq_Q$, to be the binary relation $\preceq_Q := \{(c, c') | \, c \preceq c' \text{ and } c' \in Q \Rightarrow c \in Q\}$. Observe that $\preceq_Q$ is also a preorder on P.

Let X_N be a set of *numerical* variables ranging over $\mathbb{N}$. We use $\mathcal{N}(X_N)$ to denote the set of formulae which have the members of $\{x - y \diamond c, x \diamond c \mid x, y \in X_N, c \in \mathbb{Z}, \diamond \in \{\geq, =, \leq\}\}$ as atomic formulae, and which are closed under the Boolean connectives $\neg$, $\wedge$, $\vee$. Let X_B be a finite set of *Boolean* variables. We use $\mathcal{B}(X_B)$ to denote the set of formulae which have the members of X_B as atomic formulae, and which are closed under the Boolean connectives $\neg$, $\wedge$, $\vee$. Let X' be the set of *primed* variables $\{x' \mid x \in X\}$, which refers to the "next state" values of X.

2.1 Parameterized System

Here we describe our model of parameterized systems. A simple running example of a parameterized system is given in Fig. 1. More involved examples can be found in the tech. report [3]. The example in Fig. 1 is a readers and writers protocol that uses two shared variables; A numerical variable *cnt* (the read counter) is used to keep track of the number of processes in the "read" state and a Boolean variable *lock* is used as a semaphore. The semaphore is released when the writer finished writing or all readers finished reading (*cnt* decreased to 0).

A *parameterized system* consists of an unbounded but finite number of identical processes running in parallel and operating on a finite set of shared Boolean and numerical variables. At each step, one process changes its local state and checks/updates the values of shared variables. Formally, a *parameterized system* is a triple $\mathcal{P} = (Q, T, X)$, where Q is the set of *local states*, T is the set of *transition rules*, and X is a set of shared variables. The set of shared variables X can be partitioned to the set of variables X_N ranging over $\mathbb{N}$ and X_B ranging over $\mathbb{B}$.

A transition rule $t \in T$ is of the form $\left[q \to r : \mathit{stmt}\right]$, where $q, r \in Q$ and *stmt* is a *statement* of the form $\phi_N \wedge \phi_B$, where $\phi_N \in \mathcal{N}(X_N \cup X'_N)$ and $\phi_B \in \mathcal{B}(X_B \cup X'_B)$. The formula ϕ_N controls variables ranging over $\mathbb{N}$ and ϕ_B controls Boolean variables. Taking the rule r_1 in Fig. 1 as an example, the statement says that: if the values of shared variables $\mathit{cnt} = 0$ and $\mathit{lock} = \mathit{true}$, then we are allowed to increase the value of *cnt* by 1, negate the value of *lock*, and change the local state of a process from t to r.

	shared lock: Boolean, cnt: nat
r_1:	$\left[t \to r : \mathit{cnt} = 0 \wedge \mathit{cnt}' = \mathit{cnt} + 1 \wedge \mathit{lock} \wedge \neg \mathit{lock}'\right]$
r_2:	$\left[t \to r : \mathit{cnt} >= 1 \wedge \mathit{cnt}' = \mathit{cnt} + 1\right]$
r_3:	$\left[r \to t : \mathit{cnt} >= 1 \wedge \mathit{cnt}' = \mathit{cnt} - 1\right]$
r_4:	$\left[r \to t : \mathit{cnt} = 1 \wedge \mathit{cnt}' = \mathit{cnt} - 1 \wedge \neg \mathit{lock} \wedge \mathit{lock}'\right]$
w_1:	$\left[t \to w : \mathit{lock} \wedge \neg \mathit{lock}'\right]$
w_2:	$\left[w \to t : \neg \mathit{lock} \wedge \mathit{lock}'\right]$
	Initial: t, lock

Fig. 1. Readers and writers protocol. Here t, r, w are "think", "read", and "write" states, respectively.

2.2 Transition System

A parameterized system $\mathcal{P} = (Q, T, X)$ induces an infinite-state transition system $(C, \longrightarrow)$ where C is the set of *configurations* and $\longrightarrow$ is the set of *transitions*.

A *configuration* $c \in C$ is a function $Q \cup X \to \mathbb{N} \cup \mathbb{B}$ such that (1) $c(q) \in \mathbb{N}$ gives the number of processes in state q if $q \in Q$, (2) $c(x) \in \mathbb{N}$ if $x \in X_N$ and (3) $c(x) \in \mathbb{B}$ if $x \in X_B$. We use $[x_1^{v_1}, x_2^{v_2}, \ldots, x_n^{v_n}, b_1, b_2, \ldots, b_m]$ to denote a configuration c such that (1) $c(x_i) = v_1$ for $1 \leq i \leq n$ and (2) $c(b) = \mathit{true}$ iff $b \in \{b_1, b_2, \ldots, b_m\}$.

The set of *transitions* is defined by $\longrightarrow := \bigcup_{t \in T} \xrightarrow{t}$. Let $c, c' \in C$ be two configurations and $t = \left[q \to r : \mathit{stmt}\right]$ be a transition rule. We have $(c, c') \in \xrightarrow{t}$ (written as $c \xrightarrow{t} c'$) if (1) $c'(q) = c(q) - 1$, (2) $c'(r) = c(r) + 1$, and (3) substituting each variable x in *stmt* with $c(x)$ and its primed version x' in *stmt* with $c'(x)$ produces a formula that is valid. For example, we have $\left[r^0, w^0, t^3, \mathit{cnt}^0, \mathit{lock}\right] \xrightarrow{r_1} \left[r^1, w^0, t^2, \mathit{cnt}^1\right]$ in the protocol model of Fig. 1. We use $\xrightarrow{*}$ to denote the transitive closure of $\longrightarrow$.

3 Monotonic Abstraction and CEGAR

We are interested in reachability problems, i.e., given sets of initial and bad configurations, can we reach any bad configuration from some initial configuration in the transition system induced by a given parameterized system.

We first recall the method of *monotonic abstraction* for the verification of parameterized systems and then describe an iterative and automatic CEGAR approach. The approach allows to produce more and more precise over-approximations of a given transition system from iteration to iteration. We assume a transition system $(C, \longrightarrow)$ induced by some parameterized system.

3.1 Monotonic Abstraction

Given an ordering $\trianglelefteq$ defined on C, monotonic abstraction produces an *abstract transition system* $(C, \rightsquigarrow)$ that is an over-approximation of $(C, \longrightarrow)$ and that is *monotonic* w.r.t. $\trianglelefteq$.

Definition 1 (Monotonicity). *A transition system* $(C, \rightsquigarrow)$ *is* monotonic *(w.r.t.* $\trianglelefteq$*) if for each* $c_1, c_2, c_3 \in C$*,* $c_1 \trianglelefteq c_2 \wedge c_1 \overset{t}{\rightsquigarrow} c_3 \Rightarrow \exists c_4.\ c_3 \trianglelefteq c_4 \wedge c_2 \overset{t}{\rightsquigarrow} c_4$*.*

The idea of monotonic abstraction is the following. A configuration c is allowed to use the outgoing transitions of any smaller configuration c' (w.r.t $\trianglelefteq$). The resulting system is then trivially monotonic and is an over-approximation of the original transition system. Formally, the abstract transition system $(C, \rightsquigarrow)$ is defined as follows. The set of configurations C is identical to the one of the concrete transition system. The set of *abstract transitions* is defined by $\rightsquigarrow := \bigcup_{t \in T} \overset{t}{\rightsquigarrow}$, where $(c_1, c_3) \in \overset{t}{\rightsquigarrow}$ (written as $c_1 \overset{t}{\rightsquigarrow} c_3$) iff $\exists c_2 \trianglelefteq c_1.\ c_2 \xrightarrow{t} c_3$. It is clear that $\overset{t}{\rightsquigarrow} \supseteq \xrightarrow{t}$ for all $t \in T$, i.e., $(C, \rightsquigarrow)$ over-approximates $(C, \longrightarrow)$.

In our previous works [4,6], we defined $\trianglelefteq$ to be a particular ordering $\preceq \subseteq C \times C$ such that $c \preceq c'$ iff (1) $\forall q \in Q . c(q) \leq c'(q)$, (2) $\forall n \in X_N.\ c(n) \leq c'(n)$, and (3) $\forall b \in X_B.\ c(b) = c'(b)$. Such an ordering has shown to be very useful in *shape analysis* [1] and in the verification of safety properties of *mutual exclusion* and *cache coherence* protocols [4,6]. In the CEGAR algorithm, we use $\preceq$ as the initial preorder.

3.2 Refinement of the Abstraction

Figure 2 gives an overview of the counterexample-guided abstraction refinement (CEGAR) algorithm. The algorithm works fully automatically and iteratively.

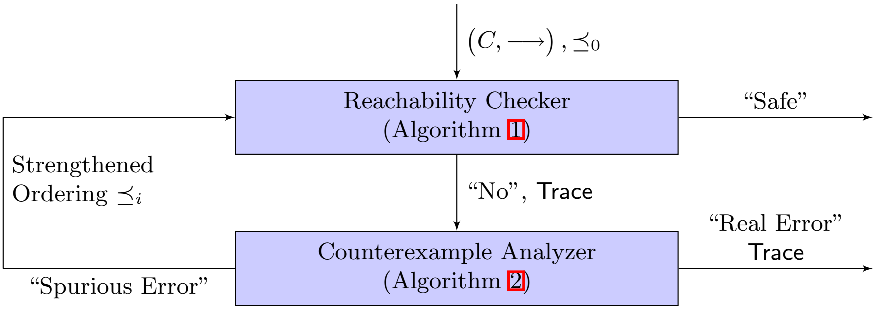

$(C, \longrightarrow)$, $\preceq_0$
Reachability Checker (Algorithm 1)
"Safe"
"No", Trace
Strengthened Ordering $\preceq_i$
"Real Error" Trace
Counterexample Analyzer (Algorithm 2)
"Spurious Error"

Fig. 2. An overview of the CEGAR algorithm (Algorithm 3)

In the beginning, a transition system $(C, \longrightarrow)$ and an initial preorder $\preceq_0$ (which equals the preorder $\preceq$ defined in the previous subsection) are given. The CEGAR algorithm (Algorithm 3) consists of two main modules, the *reachability checker* (Algorithm 1) and the *counterexample analyzer* (Algorithm 2). In the i-th iteration of the CEGAR algorithm, the *reachability checker* tests if bad configurations are reachable in the abstract transition system obtained from monotonic abstraction with the preorder $\preceq_{i-1}$. In case bad configurations are reachable, a *counterexample* is sent to the *counterexample analyzer*, which reports either "Real Error" or "Spurious Error". The latter comes with a strengthened order $\preceq_i$ (i.e., $\preceq_i \subset \preceq_{i-1}$). The strengthened order $\preceq_i$ will then be used in the $(i+1)$-th iteration of the CEGAR loop. Below we describe informally how $\preceq_{i-1}$ is strengthened to $\preceq_i$. The formal details are given in Section 4.

Strengthening the Preorder. As an example, we demonstrate using the protocol of Fig. 1 how to obtain $\preceq_1$ from $\preceq_0$. The set of bad configurations $Bad = \{c \mid c(r) \geq 1 \wedge c(w) \geq 1\}$ contains all configurations with at least one process in the "write" state and one process in the "read" state. The set of initial configurations $Init = \{c \mid c(w) = c(r) = c(cnt) = 0 \wedge c(lock)\}$ contains all configurations where all processes are in the "think" state, the value of the "cnt" equals 0, and the "lock" is available.

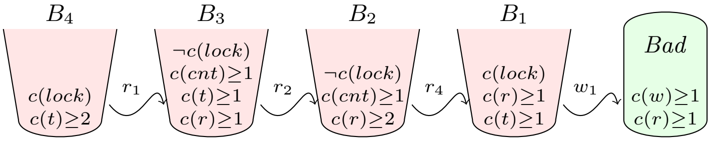


Fig. 3. The counterexample produced by backward reachability analysis on the readers and writers protocol. Notice that in the counterexample, $Init \cap B_4 \neq \emptyset$.

In iteration 1 of the CEGAR algorithm, the *reachability checker* produces a counterexample (described in Fig. 3) and sends it to the *counterexample analyzer*. More precisely, the *reachability checker* starts from the set Bad and finds the set B_1 contains all configurations that have (abstract) transitions $\overset{w_1}{\rightsquigarrow}$ to the set Bad. That is, each configuration in B_1 either has a concrete transition $\xrightarrow{w_1}$ to Bad or has some smaller configuration (w.r.t $\preceq_0$) with a concrete transition $\xrightarrow{w_1}$ to Bad. It then continues the search from B_1 and finds the set B_2 that have (abstract) transitions in $\overset{r_4}{\rightsquigarrow}$ to B_1. The sets B_3 and B_4 can be found in a similar way. It stops when B_4 is found, since $B_4 \cap Init \neq \emptyset$.

The *counterexample analyzer* simulates the received counterexample in the concrete transition system. We illustrate this scenario in Fig. 4. It starts from the set of configuration $F_4 = Init \cap B_4$[1] and checks if any bad configurations can be reached following a sequence of transitions $\xrightarrow{r_1};\xrightarrow{r_2};\xrightarrow{r_4};\xrightarrow{w_1}$. Starting from F_4, it

[1] The set of initial configurations that can reach bad configurations follows the sequence of transitions $\overset{r_1}{\rightsquigarrow};\overset{r_2}{\rightsquigarrow};\overset{r_4}{\rightsquigarrow};\overset{w_1}{\rightsquigarrow}$ in the abstract transition system

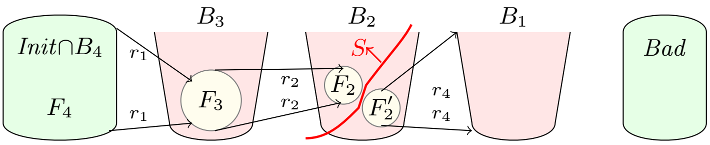


Fig. 4. Simulating the counterexample on the concrete system. Here $F_4 = Init \cap B_4 = \{c \mid c(t) \geq 2 \wedge c(w) = c(r) = c(cnt) = 0 \wedge c(lock)\}$, $F_3 = \{c \mid c(t) \geq 1 \wedge c(w) = 0 \wedge c(r) = c(cnt) = 1 \wedge \neg c(lock)\}$, $F_2 = \{c \mid c(cnt) = c(r) = 2 \wedge c(w) = 0 \wedge \neg c(lock)\}$, and $F_2' = \{c \mid c(cnt) = 1 \wedge c(r) \geq 1 \wedge \neg c(lock)\}$

finds the set F_3 which is a subset of B_3 and which can be reached from F_4 via the transition $\xrightarrow{r_1}$. It continues from F_3 and then finds the set F_2 in a similar manner via the transition $\xrightarrow{r_2}$. However, there exists no transition $\xrightarrow{r_4}$ starting from any configuration in $F_2 = \{c \mid c(cnt) = c(r) = 2 \wedge c(w) = 0 \wedge \neg c(lock)\}$. Hence the simulation stops here and concludes that the counterexample is *spurious*.

In the abstract transition system, all configurations in F_2 are able to reach B_1 via transition $\overset{r_4}{\rightsquigarrow}$ and from which they can reach *Bad* via transition $\overset{w_1}{\rightsquigarrow}$. Notice that there exists no concrete transition $\xrightarrow{r_4}$ from F_2 to B_1, but the abstract transition $\overset{r_4}{\rightsquigarrow}$ from F_2 to B_1 does exist. The reason is that all configurations in F_2 have some smaller configuration (w.r.t. $\preceq_0$) with a transition $\xrightarrow{r_4}$ to B_1. Let F_2' be the set of configurations that indeed have some transition $\xrightarrow{r_4}$ to B_1. It is clear that F_2 and F_2' are disjoint.

Therefore, we can remove the spurious counterexample by preventing configurations in F_2 from falling to some configuration in F_2' (thus also preventing them from reaching B_1). This can be achieved by first defining a set of configurations S called a "Safety Zone"with $F_2 \subseteq S$ and $F_2' \cap S = \emptyset$ and then use it to *strengthen* the preorder $\preceq_0$, i.e., let $\preceq_1 := \{(c, c') | \; c \preceq_0 c' \text{ and } c' \in S \Rightarrow c \in S\}$. In Section 4, we will explain how to use interpolation techniques [28,26] in order to automatically obtain a "Safety Zone"from a counterexample.

4 The Algorithm

In this section, we describe our CEGAR algorithm for monotonic abstraction. First, we define some concepts that will be used in the algorithm. Then, we explain the two main modules, *reachability checker* and *counterexample analyzer*. The *reachability checker* (Algorithm 1) is the backward reachability analysis algorithm on monotonic systems [2], which is possible to apply since the abstraction induces a monotonic transition system. The *counterexample analyzer* (Algorithm 2) checks a counterexample and extracts a "Safety Zone"from the counterexample if it is spurious. The CEGAR algorithm (Algorithm 3) is obtained by composing the above two algorithms. In the rest of the section, we assume a parameterized system $\mathcal{P} = (Q, T, X)$ that induces a transition system $(C, \longrightarrow)$.

4.1 Definitions

A *substitution* is a set $\{x_1 \leftarrow e_1, x_2 \leftarrow e_2, \ldots, x_n \leftarrow e_n\}$ of pairs, where x_i is a variable and e_i is a variable or a value of the same type as x_i for all $1 \leq i \leq n$. We assume that all variables are distinct, i.e., $x_i \neq x_j$ if $i \neq j$. For a formula θ and a substitution S, we use $\theta[S]$ to denote the formula obtained from θ by simultaneously replacing all free occurrences of x_i by e_i for all $x_i \leftarrow e_i \in S$. For example, if $\theta = (x_1 > x_3) \wedge (x_2 + x_3 \leq 10)$, then $\theta[x_1 \leftarrow y_1, x_2 \leftarrow 3, x_3 \leftarrow y_2] = (y_1 > y_2) \wedge (3 + y_2 \leq 10)$.

Below we define the concept of a *constraint*, a symbolic representation of configurations which we used in our algorithm. In this section, we define a number of operations on constraints. In Section 5, we show how to compute those operations.

We use $Q^{\#}$ to denote the set $\{q^{\#} \mid q \in Q\}$ of variables ranging over $\mathbb{N}$ in which each variable $q^{\#}$ is used to denote the number of processes in the state q. Define the set of formulae $\Phi := \{\phi_N \wedge \phi_B \mid \phi_N \in \mathcal{N}(Q^{\#} \cup X_N), \phi_B \in \mathcal{B}(X_B)\}$ such that each formula in Φ is a *constraint* that characterizes a potentially infinite set of configurations. Let ϕ be a constraint and c be a configuration. We write $c \vDash \phi$ if $\phi[\{q^{\#} \leftarrow c(q) \mid q \in Q\}][\{x \leftarrow c(x) \mid x \in X_N\}][\{b \leftarrow c(b) \mid b \in X_B\}]$ is a valid formula. We define the set of configurations characterized by ϕ as $[\![\phi]\!] := \{c \mid c \in C \wedge c \vDash \phi\}$. We define an *entailment relation* $\sqsubseteq$ on constraints, where $\phi_1 \sqsubseteq \phi_2$ iff $[\![\phi_1]\!] \subseteq [\![\phi_2]\!]$. We assume that the set of initial configurations *Init* and bad configurations *Bad* can be characterized by constraints ϕ_{Init} and ϕ_{Bad}, respectively.

For a constraint ϕ, the function $\mathsf{Pre}_t(\phi)$ returns a constraint characterizing the set $\{c \mid \exists c' \in [\![\phi]\!] \wedge c \xrightarrow{t} c'\}$, i.e., the set of configurations from which we can reach a configuration in $[\![\phi]\!]$ via transitions in $\xrightarrow{t}$; and $\mathsf{Post}_t(\phi)$ returns a constraint characterizing the set $\{c \mid \exists c' \in [\![\phi]\!] \wedge c' \xrightarrow{t} c\}$, i.e., the set of configurations that can be reached from some configuration in $[\![\phi]\!]$ via transitions in $\xrightarrow{t}$. For a constraint ϕ and a preorder $\preceq$ on the set of configurations, the function $\mathsf{Up}_{\preceq}(\phi)$ returns a constraint such that $[\![\mathsf{Up}_{\preceq}(\phi)]\!] = \{c' \mid \exists c \in [\![\phi]\!] \wedge c \preceq c'\}$, i.e., the upward closure of $[\![\phi]\!]$ w.r.t. the ordering $\preceq$. A *trace* (from ϕ_1 to ϕ_{n+1}) in the abstract transition system induced by monotonic abstraction and the preorder $\preceq$ is a sequence $\phi_1; t_1; \ldots; \phi_n; t_n; \phi_{n+1}$, where $\phi_i = \mathsf{Up}_{\preceq}(\mathsf{Pre}_{t_i}(\phi_{i+1}))$ and $t_i \in T$ for all $1 \leq i \leq n$. A *counterexample* (w.r.t. $\preceq$) is a trace $\phi_1; t_1; \ldots; \phi_n; t_n; \phi_{n+1}$ with $[\![\phi_1]\!] \cap [\![\phi_{Init}]\!] \neq \emptyset$ and $\phi_{n+1} = \phi_{Bad}$.

We use $\mathsf{Var}(\phi)$ to denote the set of variables that appear in the constraint ϕ. Given two constraints ϕ_A and ϕ_B such that $\phi_A \wedge \phi_B$ is unsatisfiable. An *interpolant* ϕ of (ϕ_A, ϕ_B) (denoted as $\mathsf{ITP}(\phi_A, \phi_B)$) is a formula that satisfies (1) $\phi_A \implies \phi$, (2) $\phi \wedge \phi_B$ is unsatisfiable, and (3) $\mathsf{Var}(\phi) \subseteq \mathsf{Var}(\phi_A) \cap \mathsf{Var}(\phi_B)$. Such an interpolant can be automatically found, e.g., using off-the-shelf interpolant solvers such as FOCI [28] and CLP-prover [29]. In particular, since $\phi_A, \phi_B \in \Phi$, if we use the "split solver" algorithm equipped with theory of difference bound [26] to compute an interpolant, the result will always be a formula in Φ (i.e., a constraint).

4.2 The Reachability Checker

Algorithm 1. The reachability checker

```
input  : A preorder $\preceq$ over configurations, constraints $\phi_{Init}$ and $\phi_{Bad}$
output: "Safe" or "No" with a counterexample $\phi_1; t_1; \ldots; \phi_n; t_n; \phi_{Bad}$
1 $\mathsf{Next} := \{(\phi_{Bad}, \phi_{Bad})\}$, $\mathsf{Processed} := \{\}$;
2 while Next is not empty do
3     Pick and remove a pair $(\phi_{Cur}, \mathsf{Trace})$ from Next and add it to Processed;
4     if $[\![\phi_{Cur} \wedge \phi_{Init}]\!] \neq \emptyset$ then return "No", Trace;
5     foreach $t \in T$ do
6         $\phi_{Pre} = \mathsf{Up}_{\preceq}(\mathsf{Pre}_t(\phi_{Cur}))$;
7         $old = \exists(\phi, \bullet) \in \mathsf{Next} \cup \mathsf{Processed}. \phi \sqsubseteq \phi_{Pre}$;
8         if $\neg old$ then Add $(\phi_{Pre}, \phi_{Pre}; t; \mathsf{Trace})$ to Next;
9 return "Safe";
```

Let $\preceq$ be a preorder on C and $(C, \rightsquigarrow)$ be the abstract transition system induced by the parameterized system $\mathcal{P}$ and the preorder $\preceq$. Algorithm 1 checks if the set $[\![\phi_{Init}]\!]$ is backward reachable from $[\![\phi_{Bad}]\!]$ in the abstract transition system $(C, \rightsquigarrow)$. It answers "Safe" if none of the initial configurations are backward reachable. Otherwise, it answers "No". In the latter case, it returns a *counterexample* $\phi_1; t_1; \ldots; \phi_n; t_n; \phi_{Bad}$. The algorithm uses a set Next to store constraints characterizing the sets of configurations from which it will continue the backward search. Each element in Next is a pair (ϕ, Trace), where ϕ is a constraint characterizing a set of backward reachable configurations (in the abstract transition system) and Trace is a trace from ϕ to ϕ_{Bad}. Initially, the algorithm puts in Next the constraint ϕ_{Bad}, which describes the bad configurations, together with a trace contains a singleton element namely ϕ_{Bad} itself (Line 1). In each loop iteration (excepts the last one), it picks a constraint ϕ_{Cur} (together with a trace to ϕ_{Bad}) from Next (Line 3). For each transition rule $t \in T$, the algorithm finds a constraint ϕ_{Pre} characterizing the set of configurations backward reachable from $[\![\phi_{Cur}]\!]$ via $\overset{t}{\rightsquigarrow}$ (Line 6). If there exists no constraint in Next that is larger than ϕ_{Pre} (w.r.t. $\sqsubseteq$), ϕ_{Pre} (together with a trace to ϕ_{Bad}) is added to Next (Line 7).

4.3 The Counterexample Analyzer

Given a counterexample $\phi_1; t_1; \ldots; \phi_n; t_n; \phi_{n+1}$, Algorithm 2 checks whether it is spurious or not. If spurious, it returns a constraint ϕ_S that describes a "Safety Zone"that will be used to strengthen the preorder.

As we explained in Section 3, we simulate the counterexample forwardly (Line 1-6). The algorithm begins with the constraint $\phi_1 \wedge \phi_{Init}$. If the counterexample is spurious, we will find a constraint ϕ in the i-th loop iteration for some i : $1 \leq i \leq n$ such that none of the configurations in $[\![\phi]\!]$ has transition $\xrightarrow{t_i}$ to $[\![\phi_{i+1}]\!]$ (Line 3). For this case, it computes the constraint ϕ' characterizing the set of configurations with transitions $\xrightarrow{t_i}$ to $[\![\phi_{i+1}]\!]$ (Line 4) and then computes a constraint characterizing a "Safety Zone".

Algorithm 2. The counterexample analyzer.

```
  input : A counterexample $\phi_1; t_1; \ldots; \phi_n; t_n; \phi_{n+1}$
  output: "Real Error" or "Spurious Error" with a constraint $\phi_S$
1 $\phi = \phi_1 \wedge \phi_{Init}$;
2 for $i = 1$ to $n$ do
3     if $[\![\mathsf{Post}_{t_i}(\phi)]\!] = \emptyset$ then
4         $\phi' = \mathsf{Pre}_{t_i}(\phi_{i+1})$;
5         return "Spurious Error", $\mathsf{ITP}(\phi, \phi')$;
6     $\phi = \mathsf{Post}_{t_i}(\phi) \wedge \phi_{i+1}$;
7 return "Real Error";
```

As we explained in Section 3, a "Safety Zone"is a set S of configurations that satisfies (1) $[\![\phi]\!] \subseteq S$ and (2) $S \cap [\![\phi']\!] = \emptyset$. Therefore, the constraint ϕ_S characterizing the "Safety Zone"should satisfy (1) $\phi \implies \phi_S$ and (2) $\phi_S \wedge \phi'$ is not satisfiable. The *interpolant* of (ϕ, ϕ') is a natural choice of ϕ_S that satisfies the aforesaid two conditions. Hence, in this case the algorithm returns $\mathsf{ITP}(\phi, \phi')$ (Line 5). If the above case does not happen, the algorithm computes a constraint characterizing the next set of forward reachable configurations in the counterexample (Line 6) and proceeds to the next loop iteration. It returns "Real Error" (Line 7) if the above case does not happen during the forward simulation.

4.4 The CEGAR Algorithm of Monotonic Abstraction

Algorithm 3. A CEGAR algorithm for monotonic abstraction

```
  input : An initial preorder $\preceq_0$ over configurations, constraints $\phi_{Init}$ and $\phi_{Bad}$
  output: "Safe" or "Real Error" with a counterexample $\phi_1; t_1; \ldots; \phi_n; t_n; \phi_{Bad}$
1 $i = 0$;
2 while true do
3     result = $\mathsf{ReachabilityChecker}(\preceq_i, \phi_{Init}, \phi_{Bad})$;
4     if result="No", Trace then
5         type = CounterexampleAnalyzer(Trace);
6         if type="Spurious Error", $\phi_S$ then  $i = i + 1, \preceq_i := \mathsf{Str}(\preceq_{i-1}, \phi_S)$;
7         else return "Real Error", Trace
8     else return "Safe"
```

In Algorithm 3, we describe the CEGAR approach for monotonic abstraction with the initial preorder $\preceq_0$. As described in Section 3, the algorithm works iteratively. In the i-th iteration, in Line 3, we invoke the *reachability checker* (Algorithm 1) using a preorder $\preceq_{i-1}$. When a counterexample is found, the *counterexample analyzer* (Algorithm 2) is invoked to figure out if the counterexample is real (Line 8) or spurious. In the latter case, the *counterexample analyzer* generates a constraint characterizing a "Safety Zone"and from which Algorithm 3 computes a strengthened preorder $\preceq_i$ (Line 6 and 7). The function $\mathsf{Str}(\preceq_{i-1}, \phi_S)$ in Line 8 strengthens the preorder $\preceq_{i-1}$ by the set of configurations $[\![\phi_S]\!]$.

5 Constraint Operations

In this section we explain how to compute all the constraint operations used in the algorithms in Section 4. Recall that Φ denotes the set of formulae $\{\phi_N \wedge \phi_B \mid \phi_N \in \mathcal{N}(Q^{\#} \cup X_N), \phi_B \in \mathcal{B}(X_B)\}$, where each formula in Φ is a constraint representing a set of configurations. We define $\Psi := \{\phi_N \wedge \phi_B \mid \phi_N \in \mathcal{N}(Q^{\#} \cup Q^{\#'} \cup X_N \cup X'_N), \phi_B \in \mathcal{B}(X_B \cup X'_B)\}$, where each formula in Ψ defines a relation between sets of configurations. Observe that formulae in Φ and in Ψ are closed under the Boolean connectives and substitution.

Lemma 1. *[19] Both Φ and Ψ are closed under projection (existential quantification) and the projection functions are computable.*

Lemma 2. *[19] The satisfiability problem of formulae in Φ and Ψ is decidable.*

Below we explain how to preform the constraint operations used in the algorithms in Section 4. For notational simplicity, we define $\mathsf{V} := Q^{\#} \cup X_N \cup X_B$ and $\mathsf{V}' := Q^{\#'} \cup X'_N \cup X'_B$. Let ϕ be a formula in Φ (respectively, Ψ) and X a set of variables in V (respectively, $\mathsf{V} \cup \mathsf{V}'$), we use $\exists X.\ \phi$ to denote some formula ϕ' in Φ (respectively, Ψ) obtained by the quantifier elimination algorithm (Lemma 1).

Pre and Post. The transition relation $\xrightarrow{t}$ for $t = \left[q \rightarrow r : stmt\right] \in T$ can be described by the formula $\theta^t := stmt \wedge q^{\#'} = q^{\#} - 1 \wedge r^{\#'} = r^{\#} + 1$, which is in Ψ. For a constraint ϕ, $\mathsf{Pre}_t(\phi) = \exists \mathsf{V}'.\ (\theta^t \wedge \phi[\{x \leftarrow x' \mid x \in \mathsf{V}\}]) \in \Phi$ and $\mathsf{Post}_t(\phi) = (\exists \mathsf{V}.\ (\theta^t \wedge \phi))[\{x' \leftarrow x \mid x \in \mathsf{V}\}] \in \Phi$. Both functions are computable.

Entailment. Given two constraints ϕ_1 and ϕ_2, we have $\phi_1 \sqsubseteq \phi_2$ iff $\phi_1 \wedge \neg\phi_2$ is unsatisfiable, which can be automatically checked. In practice, constraints can be easily translated into disjunctions of difference bound matrices (DBM) and hence a *sufficient* condition for entailment can be checked by standard DBM operations [19].

Intersection with Initial States. Let ϕ_{Init} be a constraint characterizing the initial configurations and ϕ_B be a constraint characterizing a set of configurations. We have $[\![\phi_{Init}]\!] \cap [\![\phi_B]\!] \neq \emptyset$ iff $\phi_{Init} \wedge \phi_B$ is satisfiable.

Strengthening. Here we explain how to strengthen an ordering $\preceq$ w.r.t a constraint $\phi_S \in \Phi$, providing that $\preceq$ is expressed as a formula $\phi_{\preceq} \in \Psi$. The strengthened order can be expressed as the formula $\phi_{\preceq_S} := \phi_{\preceq} \wedge (\phi_S \vee \neg\phi_S[\{x \leftarrow x' \mid x \in \mathsf{V}\}])$. Intuitively, for two configurations c_1 and c_2, the formula says that $c_1 \preceq_S c_2$ iff $c_1 \preceq c_2$ and either c_1 is in the "Safety Zone"or c_2 is not in the "Safety Zone".

Remark 1. The initial preorder $\preceq_0$ of our algorithm can be expressed as the formula $\bigwedge_{x \in Q^{\#} \cup X_N,\ x' \in Q^{\#'} \cup X'_N} .\ x \leq x' \wedge \bigwedge_{b \in X_B,\ b' \in X'_B} .\ (b \wedge b') \vee (\neg b \wedge \neg b')$, which is in Ψ. The constraint extracted from each spurious counterexample is in Φ if the algorithm in [26] is used to compute the interpolant. Since the initial preorder is a formula in Ψ and the constraint used for strengthening is in Φ, the formula for the strengthened order is always in Ψ and computable.

Upward Closure. We assume that the ordering $\preceq$ is expressed as a formula $\phi_{\preceq} \in \Psi$ and the constraint $\phi \in \Phi$. The upward closure of ϕ w.r.t. $\preceq$ can be captured as $\mathsf{Up}_{\preceq}(\phi) := (\exists \mathsf{V}.\ (\phi \wedge \phi_{\preceq}))[\{x' \leftarrow x \mid x \in \mathsf{V}\}]$, which is in Φ.

6 Termination

In this section, we show that each loop iteration of our CEGAR algorithm terminates. We can show by Dickson's lemma [18] that the initial preorder $\preceq$ is a WQO. An ordering over configurations is a WQO iff for any infinite sequence $c_0, c_1, c_2, \ldots$ of configurations, there are i and j such that $i < j$ and $c_i \preceq c_j$. Moreover, we can show that the strengthening of a preorder also preserves WQO.

Lemma 3. *Let S be a set of configurations. If $\preceq$ is a WQO over configurations then $\preceq_S$ is also a WQO over configurations.*

If a transition system is monotonic w.r.t. a WQO over configurations, backward reachability analysis, which is essentially a fix-point calculation, terminates within a finite number of iterations [2]. The abstract transition system is monotonic. In Section 5, we show that all the constraint operations used in the algorithms are computable. Therefore, in each iteration of the CEGAR algorithm, the termination of the *reachability checker* (Algorithm 1) is guaranteed. Since the length of a counterexample is finite, the termination of the *counterexample analyzer* (Algorithm 2) is also guaranteed. Hence, we have the following lemma.

Lemma 4. *Each loop iteration of the CEGAR algorithm (Algorithm 3) is guaranteed to terminate.*

7 Extension

The model described in Section 2 can be extended to allow some additional features. For example, (1) dynamic creation of processes $\left[\cdot \rightarrow q : \mathit{stmt}\right]$, (2) dynamic deletion of processes $\left[q \rightarrow \cdot : \mathit{stmt}\right]$, and (3) synchronous movement $\left[q_1, q_2, \ldots, q_n \rightarrow r_1, r_2, \ldots, r_n : \mathit{stmt}\right]$. Moreover, the language of the *statement* can be extended to any formula in Presburger arithmetic. For all of the new features, we can use the same constraint operations as in Section 5; the extended transition rule still can be described using a formula in Ψ, Presburger arithmetic is closed under Boolean connectives, substitution, and projection and all the mentioned operations are computable.

8 Case Studies and Experimental Results

We have implemented a prototype and tested it on several case studies of classical synchronization schemes and reference counting schemes, which includes

Table 1. Summary of experiments of case studies. *Interpolant* denotes the kind of interpolant prover we use, where DBM denotes the difference bound matrix based solver, and CLP denotes the CLP-prover. *Pass* indicates whether the refinement procedure can terminate with a specific interpolant prover. *Time* is the execution time of the program, measured by the bash `time` command. *#ref* is the number of refinements needed to verify the property. *#cons* is the total number of constraints generated by the reachability checker. For each model, we use *#t*, *#l*, *#s* to denote the number of transitions, the number of local variables, and the number of shared variables, respectively. All case studies are described in details in tech. report [3].

model	**interpolant**	**pass**	**time**	**#ref**	**#cons**	**#t**	**#l**	**#s**
readers/writers	DBM	√	0.04 sec	1	90	6	5	2
	CLP	√	0.08 sec	1	90			
refined readers/writers priority to readers	DBM	√	3.9 sec	2	3037	8	5	3
	CLP	X	-	-	-			
refined readers/writers priority to writers	DBM	√	3.5 sec	1	2996	12	7	5
	CLP	√	68 sec	4	39191			
sleeping barbers	DBM	√	3.9 sec	1	1518	10	15	1
	CLP	√	4.1 sec	1	1518			
pmap reference counting	DBM	√	0.1 sec	1	249	25	4	7
	CLP	√	0.1 sec	1	249			
reference counting gc	DBM	√	0.02 sec	1	19	7	4	1
	CLP	√	0.05 sec	1	19			
missionary & cannibals	DBM	X	-	-	-	7	7	1
	CLP	√	0.1 sec	3	86			
swimming pool v2	DBM	√	0.2 sec	2	59	6	0	10
	CLP	√	0.2 sec	2	55			

readers/writers protocol, sleeping barbers problem, the missionaries/cannibals problem [11], the swimming pool protocol [11,23], and virtual memory management. These case studies make use of shared counters (in some cases protected by semaphores) to keep track of the number of current references to a given resource. Monotonic abstraction returns spurious counterexamples for all the case studies. In our experiments, we use two interpolating procedures to refine the abstraction. One is a homemade interpolant solver based on difference bound matrices [26]; the other one is the CLP-prover [29], an interpolant solvers based on constraint logic programming. The results, obtained on an Intel Xeon 2.66GHz processor with 8GB memory, are listed in Table 1. It shows that our CEGAR method efficiently verifies many examples in a completely automatic manner.

We compare our approach with three related tools: the ALV tool [14], the Interproc Analyzer [24], and FASTer [11] based on several examples (and their variants) from our case studies. The results are summarized in Table 2. Note that these tools either perform an exact forward analysis where the invariant is exactly represented (FASTer), or try to capture all possible invariants of a certain form (ALV and Interproc Analyzer). In these two approaches, the verification of the property is deduced from the sometimes expensively generated invariants. The main difference between our approach and the other ones is that

Table 2. Summary of tool comparisons. For cma, we selected the best results among the ones obtained from DBM and CLP. For FASTer, we tested our examples with library Mona and the backward search strategy. For the other tools, we just used the default settings. In our experiment, ALV outputted "unable to verify" for the missionaries/cannibals model and failed to verify the other test cases after one day of execution. FASTer failed to verify four of the six test cases within a memory limit of 8GB. Interproc Analyzer gave false positives for examples other than the swimming pool protocol and the missionaries/cannibals model. That is, it proved reachability for models where the bad states were not reachable.

Model	**Tool**	**Pass**	**Result**	**Model**	**Tool**	**Pass**	**Result**
swimming pool protocol v2	cma	√	0.2 sec	pmap reference counting	cma	√	0.1 sec
	FASTer	X	oom		FASTer	√	85 sec
	Interproc	√	2.7 sec		Interproc	X	false positive
	ALV	X	timeout		ALV	X	timeout
Model	**Tool**	**Pass**	**Result**	**Model**	**Tool**	**Pass**	**Result**
missionary & cannibals	cma	√	0.1 sec	readers writers pri. readers	cma	√	3.9 sec
	FASTer	X	oom		FASTer	√	3 min 44 sec
	Interproc	√	2 sec		Interproc	X	false positive
	ALV	X	cannot verify		ALV	X	timeout
Model	**Tool**	**Pass**	**Result**	**Model**	**Tool**	**Pass**	**Result**
missionary & cannibals v2	cma	√	0.2 sec	readers writers pri. readers v2	cma	√	0.5 sec
	FASTer	X	oom		FASTer	X	oom
	Interproc	X	false positive		Interproc	X	false positive
	ALV	X	timeout		ALV	X	timeout

we concentrate on minimal constraints to track the violation of the property at hand. Using upward closed sets as a symbolic representation efficiently exploits the monotonicity of the abstract system where the analysis is exact yet efficient.

9 Related and Future Work

We have presented a method for refining monotonic abstraction in the context of verification of safety properties for parameterized systems. We have implemented a prototype based on the method and used it to automatically verify parameterized versions of synchronization and reference counting schemes. Our method adopts an iterative counter-example guided abstraction refinement (CEGAR) scheme. Abstraction refinement algorithms for forward/backward analysis of well-structured models have been proposed in [25,16]. Our CEGAR scheme is designed instead for undecidable classes of models. Other tools dealing with the verification of similar parameterized systems can be divided into two categories: exact and approximate. In Section 8, we compare our method to a representative from each category. The results confirm the following. Exact techniques, such as FASTer [11], restrict their computations to under-approximations of the set of reachable states. They rely on computing the exact effect of particular categories of loops, like non-nested loops for instance, and may not terminate in general. On the contrary, our method is guaranteed to terminate at each iteration.On the other hand, approximate techniques like ALV and the Interproc Analyzer [14,24], rely on widening operators in order to ensure termination. Typically, such operators correspond to extrapolations that come with a loss of precision. It is unclear how to refine the obtained over-approximations when false positives appear in parameterized systems like those we study.

Also, the refinement method proposed in the present paper allows us to automatically verify new case studies (e.g. reference counting schemes) that cannot be handled by regular model checking [27,17,9,12,30,13], monotonic abstractions [6,4,7] (they give false positives), environment abstraction [15], and invisible invariants [10]. It is important to remark that a distinguished feature of our method with respect to methods like invisible invariants and environment abstraction is that we operate on abstract models that are still infinite-state thus trying to reduce the loss of precision in the approximation required to verify a property.

We currently work on extensions of our CEGAR scheme to systems in which processes are linearly ordered. Concerning this point, in [5] we have applied a manually supplied strengthening of the subword ordering to automatically verify a formulation of Szymanski's algorithm (defined for ordered processes) with non-atomic updates.

References

1. Abdulla, P.A., Bouajjani, A., Cederberg, J., Haziz, F., Rezine, A.: Monotonic abstraction for programs with dynamic memory heaps. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 341–354. Springer, Heidelberg (2008)
2. Abdulla, P.A., Čerāns, K., Jonsson, B., Tsay, Y.-K.: General decidability theorems for infinite-state systems. In: LICS 1996, pp. 313–321 (1996)
3. Abdulla, P.A., Chen, Y.-F., Delzanno, G., Haziza, F., Hong, C.-D., Rezine, A.: Constrained monotonic abstraction: a cegar for parameterized verification. Tech. report 2010-015, Uppsala University, Sweden (2010)
4. Abdulla, P.A., Delzanno, G., Rezine, A.: Parameterized verification of infinite-state processes with global conditions. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 145–157. Springer, Heidelberg (2007)
5. Abdulla, P.A., Delzanno, G., Rezine, A.: Approximated context-sensitive analysis for parameterized verification. In: FMOODS 2009/FORTE 2009, pp. 41–56 (2009)
6. Abdulla, P.A., Henda, N.B., Delzanno, G., Rezine, A.: Regular model checking without transducers (on efficient verification of parameterized systems). In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 721–736. Springer, Heidelberg (2007)
7. Abdulla, P.A., Henda, N.B., Delzanno, G., Rezine, A.: Handling parameterized systems with non-atomic global conditions. In: Logozzo, F., Peled, D.A., Zuck, L.D. (eds.) VMCAI 2008. LNCS, vol. 4905, pp. 22–36. Springer, Heidelberg (2008)
8. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Info. Comput. 127(2), 91–101 (1996)
9. Abdulla, P.A., Jonsson, B., Nilsson, M., d'Orso, J.: Regular model checking made simple and efficient. In: Brim, L., Jančar, P., Křetínský, M., Kucera, A. (eds.) CONCUR 2002. LNCS, vol. 2421, pp. 116–130. Springer, Heidelberg (2002)

10. Arons, T., Pnueli, A., Ruah, S., Xu, J., Zuck, L.: Parameterized verification with automatically computed inductive assertions. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 221–234. Springer, Heidelberg (2001)
11. Bardin, S., Leroux, J., Point, G.: FAST extended release. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 63–66. Springer, Heidelberg (2006)
12. Boigelot, B., Legay, A., Wolper, P.: Iterating transducers in the large. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 223–235. Springer, Heidelberg (2003)
13. Bouajjani, A., Habermehl, P., Vojnar, T.: Abstract regular model checking. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 372–386. Springer, Heidelberg (2004)
14. Bultan, T., Yavuz-Kahveci, T.: Action language verifier. In: ASE 2001, p. 382 (2001)
15. Clarke, E., Talupur, M., Veith, H.: Environment abstraction for parameterized verification. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 126–141. Springer, Heidelberg (2005)
16. Cousot, P., Ganty, P., Raskin, J.-F.: Fixpoint-guided abstraction refinements. LNCS. Springer, Heidelberg (2007)
17. Dams, D., Lakhnech, Y., Steffen, M.: Iterating transducers. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 286–297. Springer, Heidelberg (2001)
18. Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. Amer. J. Math. 35, 413–422 (1913)
19. Dill, D.: Timing assumptions and verification of finite-state concurrent systems. In: AVMFSS 1989, pp. 197–212 (1989)
20. Emerson, E., Namjoshi, K.: On model checking for non-deterministic infinite-state systems. In: LICS 1998, pp. 70–80 (1998)
21. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: LICS 1999 (1999)
22. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! TCS 256(1-2), 63–92 (2001)
23. Fribourg, L., Olsén, H.: Proving safety properties of infinite state systems by compilation into Presburger arithmetic. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 213–227. Springer, Heidelberg (1997)
24. Gal Lalire, M.A., Jeannet, B.: A web interface to the interproc analyzer, http://pop-art.inrialpes.fr/interproc/interprocweb.cgi
25. Geeraerts, G., Raskin, J.-F., Begin, L.V.: Expand, enlarge and check... made efficient. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 394–404. Springer, Heidelberg (2005)
26. Jhala, R., McMillan, K.L.: A practical and complete approach to predicate refinement. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 459–473. Springer, Heidelberg (2006)
27. Kesten, Y., Maler, O., Marcus, M., Pnueli, A., Shahar, E.: Symbolic model checking with rich assertional languages. TCS 256, 93–112 (2001)
28. McMillan, K.L.: An interpolating theorem prover. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 101–121. Springer, Heidelberg (2004)
29. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint solving for interpolation. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 346–362. Springer, Heidelberg (2007)
30. Touili, T.: Regular Model Checking using Widening Techniques. ENTCS 50(4) (2001)

Information Flow in Interactive Systems

Mário S. Alvim[1], Miguel E. Andrés[2], and Catuscia Palamidessi[1]

[1] INRIA and LIX, École Polytechnique Palaiseau, France
[2] Institute for Computing and Information Sciences, The Netherlands

Abstract. We consider the problem of defining the information leakage in interactive systems where secrets and observables can alternate during the computation. We show that the information-theoretic approach which interprets such systems as (simple) noisy channels is not valid anymore. However, the principle can be recovered if we consider more complicated types of channels, that in Information Theory are known as channels with memory and feedback. We show that there is a complete correspondence between interactive systems and such kind of channels. Furthermore, we show that the capacity of the channels associated to such systems is a continuous function of the Kantorovich metric.

1 Introduction

Information leakage refers to the problem that the observable parts of the behavior of a system may reveal information that we would like to keep secret. In recent years, there has been a growing interest in the quantitative aspects of this problem, partly because it is convenient to represent the partial knowledge of the secrets as a probability distribution, and partly because the mechanisms to protect the information may use randomization to obfuscate the relation between the secrets and the observables.

Among the quantitative approaches, some of the most popular ones are based on Information Theory [5,12,4,16]. The system is interpreted as an information-theoretic *channel*, where the secrets are the input and the observables are the output. The channel matrix is constituted by the conditional probabilities $p(b \mid a)$, defined as the measure of the executions that give observable b within those which contain the secret a. The leakage is represented by the *mutual information*, and the worst-case leakage by the *capacity* of the channel.

In the above works, the secret value is assumed to be chosen at the beginning of the computation. In this paper, we are interested in *Interactive systems*, i.e. systems in which secrets and observables can alternate during the computation, and influence each other. Examples of interactive protocols include *auction protocols* like [21,18,17]. Some of these have become very popular thanks to their integration in Internet-based electronic commerce platforms [9,10,14]. As for interactive programs, examples include web servers, GUI applications, and command-line programs [3].

We investigate the applicability of the information-theoretic approach to interactive systems. In [8] it was proposed to define the matrix elements $p(b \mid a)$ as the measure of the traces with (secret, observable)-projection (a, b), divided by the measure of the trace with secret projection a. This follows the definition of conditional probability in terms of joint and marginal probability. However, it does not define an information-theoretic

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 102–116, 2010.

channel. In fact, by definition a channel should be invariant with respect to the input distribution, and such construction is not, as shown by the following example.

Example 1. Figure 1 represents a web-based interaction between one seller and two possible buyers, *rich* and *poor*. The seller offers two different products, *cheap* and *expensive*, with given probabilities. Once the product is offered, each buyer may try to buy the product, with a certain probability. For simplicity we assume that the buyers offers are exclusive. We assume that the offers are observables, in the sense that they are made public in the website, while the identity of the buyer that actually buys the product should be secret to an external observer. The symbols r, s, t, $\overline{r}$, $\overline{s}$, $\overline{t}$ represent the probabilities, with the convention that $\overline{r} = 1 - r$.

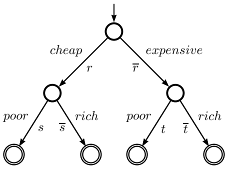


Fig. 1. Inter. System

Following [8] we can compute the conditional probabilities as $p(b|a) = \frac{p(a,b)}{p(a)}$, thus obtaining the matrix on Table 1.

However, the matrix is not invariant with respect to the input distribution. For instance, if we fix $r = \overline{r} = 0.5$ and consider two different input distributions, obtained by varying the values of (s, t), *we get two different matrices of conditional probabilities*, which are represented in Table 2. Hence when the secrets occur *after* the observables we cannot consider the conditional probabilities as representing a (classical) channel, and we cannot apply the standard information-theoretic concepts. In particular, we cannot adopt the (classical) capacity to represent the worst-case leakage, since the capacity is defined using a fixed channel matrix over all possible input distributions.

The first contribution of this paper is to consider an extension of the theory of channels which makes the information-theoretic approach applicable also the case of interactive systems. It turns out that a richer notion of channels, known in Information Theory as *channels with memory and feedback*, serves our purposes. The dependence of inputs on previous outputs corresponds to feedback, and the dependence of outputs on previous inputs and outputs corresponds to memory.

Table 1. Cond. probabilities of Example 1

	cheap	*expensive*
poor	$\frac{rs}{rs+\overline{r}t}$	$\frac{\overline{r}t}{rs+\overline{r}t}$
rich	$\frac{r\overline{s}}{r\overline{s}+\overline{r}\overline{t}}$	$\frac{\overline{r}\overline{t}}{r\overline{s}+\overline{r}\overline{t}}$

Table 2. Two different channel matrices induced by two different input distributions

	cheap	*expensive*	Input dist.
poor	$\frac{2}{5}$	$\frac{3}{5}$	$p(poor) = \frac{1}{2}$
rich	$\frac{3}{5}$	$\frac{2}{5}$	$p(rich) = \frac{1}{2}$

(a) $r = \frac{1}{2}, s = \frac{2}{5}, t = \frac{3}{5}$

	cheap	*expensive*	Input dist.
poor	$\frac{1}{4}$	$\frac{3}{4}$	$p(poor) = \frac{1}{5}$
rich	$\frac{9}{16}$	$\frac{7}{16}$	$p(rich) = \frac{4}{5}$

(b) $r = \frac{1}{2}, s = \frac{1}{10}, t = \frac{3}{10}$

A second contribution of our work is the proof that the channel capacity is a continuous function of the Kantorovich metric on interactive systems. This was pointed out also in [8], however their construction does not work in our case due to the fact that (as far as we understand) it assumes that the probability of a secret action, in any point of the computation, is not 0. This assumption is not guaranteed in our case and therefore we had to proceed differently.

A more complete version of this paper (with proofs) is on line [1].

2 Preliminaries

2.1 Concepts from Information Theory

For more detailed information on this part we refer to [6]. Let A, B denote two random variables with corresponding probability distributions $p_A(\cdot), p_B(\cdot)$, respectively. We shall omit the subscripts when they are clear from the context. Let $\mathcal{A} = \{a_1, \ldots, a_n\}$, $\mathcal{B} = \{b_1, \ldots, b_m\}$ denote, respectively, the sets of possible values for A and for B.

The *entropy* of A is defined as $H(A) = -\sum_{\mathcal{A}} p(a_i) \log p(a_i)$ and it measures the uncertainty of A. It takes its minimum value $H(A) = 0$ when $p_A(\cdot)$ is a delta of Dirac. The maximum value $H(A) = \log |\mathcal{A}|$ is obtained when $p_A(\cdot)$ is the uniform distribution. Usually the base of the logarithm is set to be 2 and the entropy is measured in *bits*. The *conditional entropy* of A given B is $H(A|B) = -\sum_{\mathcal{B}} p(b_i) \sum_{\mathcal{A}} p(a_j|b_i) \log p(a_j|b_i)$, and it measures the uncertainty of A when B is known. We can prove that $0 \leq H(A|B) \leq H(A)$. The minimum value, 0, is obtained when A is completely determined by B. The maximum value $H(A)$ is obtained when A and B are independent. The *mutual information* between A and B is defined as $I(A;B) = H(A) - H(A|B)$, and it measures the amount of information about A that we gain by observing B. It can be shown that $I(A;B) = I(B;A)$ and $0 \leq I(A;B) \leq H(A)$.

The entropy and mutual information respect the *chain laws*. Namely, given a sequence of random variables $A_1, A_2, \ldots, A_k$ and B, we have:

$$H(A_1, A_2, \ldots, A_k) = \sum_{i=1}^{k} H(A_i|A_1, \ldots, A_{i-1}) \tag{1}$$

$$I(A_1, A_2, \ldots, A_k; B) = \sum_{i=1}^{k} I(A_i; B|A_1, \ldots, A_{i-1}) \tag{2}$$

A (*discrete memoryless*) *channel* is a tuple $(\mathcal{A}, \mathcal{B}, p(\cdot|\cdot))$, where $\mathcal{A}, \mathcal{B}$ are the sets of input and output symbols, respectively, and $p(b_j|a_i)$ is the probability of observing the output symbol b_j when the input symbol is a_i. An input distribution $p(a_i)$ over $\mathcal{A}$ determines, together with the channel, the joint distribution $p(a_i, b_j) = p(a_i|b_j) \cdot p(a_i)$ and consequently $I(A;B)$. The maximum $I(A;B)$ over all possible input distributions is the channel's *capacity*. Shannon's famous result states that the capacity coincides with the maximum rate by which information can be transmitted using the channel.

In this paper we consider input and output *sequences* instead of just symbols.

Convention 1. *Let $\mathcal{A} = \{a_1, \ldots, a_n\}$ be a finite set of n different symbols (*alphabet*). When we have a sequence of symbols (ordered in time), we use a Greek letter α_t to denote the symbol at time t. The notation α^t stands for the sequence $\alpha_1 \alpha_2 \ldots \alpha_t$. For instance, in the sequence $a_3 a_7 a_5$, we have $\alpha_2 = a_7$ and $\alpha^2 = a_3 a_7$.*

Convention 2. *Let X be a random variable. X^t denotes the sequence of t consecutive occurrences $X_1, \ldots, X_t$ of the random variable X.*

When the channel is used repeatedly, the discrete memoryless channel described above represents the case in which the behavior of the channel at the present time does not depend upon the past history of inputs and outputs. If this assumption does not hold, then we have a channel *with memory*. Furthermore, if the outputs from the channel can be fed back to the encoder, thus influencing the generation of the next input symbol, then the channel is said to be *with feedback*; otherwise it is *without feedback*.

Equation 3 makes explicit the probabilistic behavior of channels regarding those classifications. Suppose a general channel from $\mathcal{A}$ to $\mathcal{B}$ with the associated random variables A for input and B for output. Using the notation introduced in Convention 1, the channel behavior after T uses can be fully described by the joint probability $p(\alpha^T, \beta^T)$.

Using probability laws we derive:

$$p(\alpha^T, \beta^T) = \prod_{t=1}^{T} p(\alpha_t | \alpha^{t-1}, \beta^{t-1}) p(\beta_t | \alpha^t, \beta^{t-1}) \qquad \text{(by the expansion law)} \quad (3)$$

The first term $p(\alpha_t|\alpha^{t-1}, \beta^{t-1})$ indicates that the probability of α_t depends not only on α^{t-1}, but also on β^{t-1} (*feedback*). The second term $p(\beta_t|\alpha^t, \beta^{t-1})$ indicates that the probability of each β_t depends on previous history of inputs α^t and outputs β^{t-1} (*memory*).

If the channel is without feedback, then we have that $p(\alpha_t|\alpha^{t-1}, \beta^{t-1}) = p(\alpha_t|\alpha^{t-1})$, and if the channel is without memory, then we have also $p(\beta_t|\alpha^t, \beta^{t-1}) = p(\beta_t|\alpha_t)$. From these we derive $p(\beta^T|\alpha^T) = \prod_{t=1}^{T} p(\beta_t|\alpha_t)$, which is the classic equation for discrete memoryless channels without feedback.

Let $(\mathcal{V}, \mathcal{K})$ be a Borel space and let $(\mathcal{X}, \mathcal{B}_{\mathcal{X}})$ and $(\mathcal{Y}, \mathcal{B}_{\mathcal{Y}})$ be Polish spaces equipped with their Borel σ-algebras. Let $\rho(\mathrm{d}x|v)$ be a family of measures on $\mathcal{X}$ given $\mathcal{V}$. Then $\rho(\mathrm{d}x|v)$ is a *stochastic kernel* if and only if and only if $\rho(\cdot|v)$ is a random variable from $\mathcal{V}$ into the power set $\mathcal{P}(\mathcal{X})$.

2.2 Probabilistic Automata

A function $\mu \colon \mathcal{S} \rightarrow [0, 1]$ is a *discrete probability distribution* on a countable set $\mathcal{S}$ if $\sum_{s \in \mathcal{S}} \mu(s) = 1$ and $\mu(s) \geq 0$ for all s. The set of all discrete probability distributions on $\mathcal{S}$ is $\mathcal{D}(\mathcal{S})$.

A *probabilistic automaton* [15] is a quadruple $M = (\mathcal{S}, \mathcal{L}, \hat{s}, \vartheta)$ where $\mathcal{S}$ is a countable set of *states*, $\mathcal{L}$ a finite set of *labels* or *actions*, $\hat{s}$ the *initial* state, and ϑ a *transition function* $\vartheta : \mathcal{S} \rightarrow \wp_f(\mathcal{D}(\mathcal{L} \times \mathcal{S}))$. Here $\wp_f(X)$ is the set of all finite subsets of X. If $\vartheta(s) = \emptyset$ then s is a *terminal* state. We write $s{\rightarrow}\mu$ for $\mu \in \vartheta(s),\ s \in \mathcal{S}$. Moreover, we write $s \xrightarrow{\ell} r$ for $s, r \in \mathcal{S}$ whenever $s{\rightarrow}\mu$ and $\mu(\ell, r) > 0$. A *fully probabilistic automaton* is a probabilistic automaton satisfying $|\vartheta(s)| \leq 1$ for all states. When $\vartheta(s) \neq \emptyset$ we overload the notation and denote $\vartheta(s)$ the distribution outgoing from s.

A *path* in a probabilistic automaton is a sequence $\sigma = s_0 \xrightarrow{\ell_1} s_1 \xrightarrow{\ell_2} \cdots$ where $s_i \in \mathcal{S}$, $\ell_i \in \mathcal{L}$ and $s_i \xrightarrow{\ell_{i+1}} s_{i+1}$. A path can be *finite* in which case it ends with a state. A path is *complete* if it is either infinite or finite ending in a terminal state. Given a finite path σ, $\mathrm{last}(\sigma)$ denotes its last state. Let $\mathrm{Paths}_s(M)$ denote the set of all paths, $\mathrm{Paths}^\star{}_s(M)$ the set of all finite paths, and $\mathrm{CPaths}_s(M)$ the set of all complete paths of an automaton M, starting from the state s. We will omit s if $s = \hat{s}$. Paths are ordered by the prefix relation, which we denote by $\leq$. The *trace* of a path is the sequence of actions in $\mathcal{L}^* \cup \mathcal{L}^\infty$ obtained by removing the states, hence for the above σ we have $\mathit{trace}(\sigma) = l_1 l_2 \ldots$. If $\mathcal{L}' \subseteq \mathcal{L}$, then $\mathit{trace}_{\mathcal{L}'}(\sigma)$ is the projection of $\mathit{trace}(\sigma)$ on the elements of $\mathcal{L}'$.

Let $M = (\mathcal{S}, \mathcal{L}, \hat{s}, \vartheta)$ be a (fully) probabilistic automaton, $s \in \mathcal{S}$ a state, and let $\sigma \in \mathrm{Paths}^\star_s(M)$ be a finite path starting in s. The *cone* generated by σ is the set of complete paths $\langle\sigma\rangle = \{\sigma' \in \mathrm{CPaths}_s(M) \mid \sigma \leq \sigma'\}$. Given a fully probabilistic automaton $M = (\mathcal{S}, \mathcal{L}, \hat{s}, \vartheta)$ and a state s, we can calculate the *probability value*, denoted by $\mathbf{P}_s(\sigma)$, of any finite path σ starting in s as follows: $\mathbf{P}_s(s) = 1$ and $\mathbf{P}_s(\sigma \xrightarrow{\ell} s') = \mathbf{P}_s(\sigma)\ \mu(\ell, s')$, where $\mathrm{last}(\sigma) \rightarrow \mu$.

Let $\Omega_s \triangleq \mathrm{CPaths}_s(M)$ be the sample space, and let $\mathcal{F}_s$ be the smallest σ-algebra generated by the cones. Then $\mathbf{P}$ induces a unique *probability measure* on $\mathcal{F}_s$ (which we will also denote by $\mathbf{P}_s$) such that $\mathbf{P}_s(\langle\sigma\rangle) = \mathbf{P}_s(\sigma)$ for every finite path σ starting in s. For $s = \hat{s}$ we write $\mathbf{P}$ instead of $\mathbf{P}_{\hat{s}}$.

Given a probability space $(\Omega, \mathcal{F}, P)$ and two events $A, B \in \mathcal{F}$ with $P(B) > 0$, the *conditional probability* of A given B, $P(A \mid B)$, is defined as $P(A \cap B)/P(B)$.

3 Discrete Channels with Memory and Feedback

We adopt the model proposed in [19] for discrete channels with memory and feedback. Such model, represented in Figure 2, can be decomposed in sequential components as follows. At time t the internal channel's behavior is represented by the conditional probabilities $p(\beta_t|\alpha^t, \beta^{t-1})$. The internal channel takes the input α_t and, according to the history of inputs and outputs up to the moment α^t, β^{t-1}, produces an output symbol β_t. The output is then fed back to the encoder with delay one. On the other side, at time t the encoder takes the message and the past output symbols β^{t-1}, and produces a channel input symbol α_t. At final time T the decoder takes all the channel outputs β^T and produces the decoded message $\hat{W}$. The order is the following:

$$\text{Message } W, \quad \alpha_1, \beta_1, \quad \alpha_2, \beta_2, \quad \ldots, \quad \alpha_T, \beta_T, \quad \text{Decoded Message } \hat{W}$$

Let us describe such channel in more detail. Let $\mathcal{A}$ and $\mathcal{B}$ be two finite sets. Let $\{A_t\}_{t=1}^T$ (channel's input) and $\{B_t\}_{t=1}^T$ (channel's output) be families of random variables in $\mathcal{A}$ and $\mathcal{B}$ respectively. Moreover, let $\mathcal{A}^T$ and $\mathcal{B}^T$ represent their T-fold product spaces. A *channel* is a family of stochastic kernels $\{p(\beta_t|\alpha^t, \beta^{t-1})\}_{t=1}^T$.

Let $\mathcal{F}_t$ be the set of all measurable maps $\varphi_t : \mathcal{B}^{t-1} \rightarrow \mathcal{A}$ endowed with a probability distribution, and let F_t be the corresponding random variable. Let $\mathcal{F}^T$, F^T denote the Cartesian product on the domain and the random variable, respectively. A *channel code function* is an element $\varphi^T = (\varphi_1, \ldots, \varphi_T) \in \mathcal{F}^T$.

Note that, by probability laws, $p(\varphi^T) = \prod_{t=1}^T p(\varphi_t|\varphi^{t-1})$. Hence the distribution on $\mathcal{F}^T$ is uniquely determined by a sequence $\{p(\varphi_t|\varphi^{t-1})\}_{t=1}^T$. We will use the notation $\varphi^t(\beta^{t-1})$ to represent the $\mathcal{A}$-valued t-tuple $(\varphi_1, \varphi_2(\beta^1), \ldots, \varphi_t(\beta^{t-1}))$.

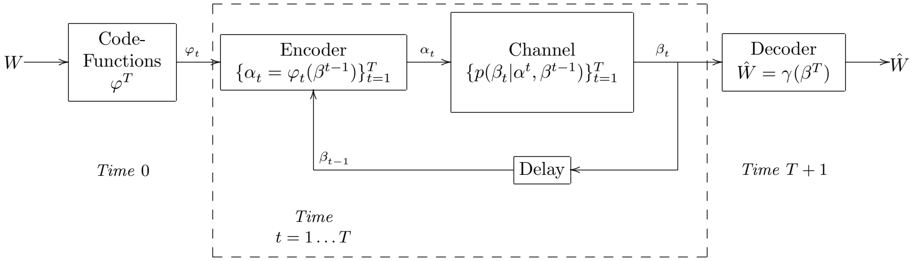

W
Code-Functions φ^T
φ_t
Encoder $\{\alpha_t = \varphi_t(\beta^{t-1})\}_{t=1}^T$
α_t
Channel $\{p(\beta_t|\alpha^t, \beta^{t-1})\}_{t=1}^T$
β_t
Decoder $\hat{W} = \gamma(\beta^T)$
$\hat{W}$
β_{t-1}
Delay
Time 0
Time $T+1$
Time $t = 1 \dots T$

Fig. 2. Model for discrete channel with memory and feedback

In Information Theory this kind of channels are used to encode and transmit messages. If $\mathcal{W}$ is a message set of cardinality M with typical element w, endowed with a probability distribution, a *channel code* is a set of M channel code functions $\varphi^T[w]$, interpreted as follows: for message w, if at time t the channel feedback is β^{t-1}, then the channel encoder outputs $\varphi_t[w](\beta^{t-1})$. A *channel decoder* is a map from $\mathcal{B}^T$ to $\mathcal{W}$ which attempts to reconstruct the input message after observing all the output history β^T from the channel.

3.1 Directed Information and Capacity of Channels with Feedback

In classical Information Theory, the channel capacity, which is related to the channel's transmission rate by Shannon's fundamental result, can be obtained as the supremum of the mutual information over all possible input's distributions. In presence of feedback, however, this correspondence does not hold anymore. More specifically, mutual information does not represent any longer the information flow from α^T to β^T. Intuitively, this is due to the fact that mutual information expresses correlation, and therefore it is increased by feedback. But the feedback, i.e the way the output influences the next input, is part of the a priori knowledge, and therefore should not be counted when we measure the output's contribution to the reduction of the uncertainty about the input. If we want to maintain the correspondence with the transmission rate and with information flow, we need to replace mutual information with *directed information* [13].

Definition 1. *In a channel with feedback, the directed information from input* A^T *to output* B^T *is defined as* $I(A^T \rightarrow B^T) = \sum_{t=1}^T I(\alpha^t; \beta_t|\beta^{t-1})$. *In the other direction, the directed information from* B^T *to* A^T *is defined as:* $I(B^T \rightarrow A^T) = \sum_{t=1}^T I(\alpha_t; \beta^{t-1}|\alpha^{t-1})$.

Note that the directed information defined above are not symmetric: the flow from A^T to B^T takes into account the correlation between α^t and β_t, while the flow from B^T to A^T is based on the correlation between β^{t-1} and α_t . Intuitively, this is because α^t influences β_t, but, in the other direction, it is β^{t-1} that influences α_t.

It can be proved [19] that $I(A^T; B^T) = I(A^T \rightarrow B^T) + I(B^T \rightarrow A^T)$. If a channel does not have feedback, then $I(B^T \rightarrow A^T) = 0$ and $I(A^T; B^T) = I(A^T \rightarrow B^T)$.

In a channel with feedback the information transmitted is the directed information, and not the mutual information. The following example should help understanding why.

Example 2. Consider the discrete memoryless channel with input alphabet $\mathcal{A} = \{a_1, a_2\}$ and output alphabet $\mathcal{B} = \{b_1, b_2\}$ whose matrix is represented in Table 3.

Suppose that the channel is used with feedback, in such a way that, for all t's, $\alpha_{t+1} = a_1$ if $\beta_t = b_1$, and $\alpha_{t+1} = a_2$ if $\beta_t = b_2$. It is easy to show that if $t \geq 2$ then $I(A^t; B^t) \neq 0$. However, there is no leakage from from A^t to B^t, since the rows of the matrix are all equal. We have indeed that $I(A^t \rightarrow B^t) = 0$, and the mutual information $I(A^t; B^t)$ is only due to the feedback information flow $I(B^t \rightarrow A^t)$.

Table 3. Channel matrix for Example 2

	b_1	b_2
a_1	0.5	0.5
a_2	0.5	0.5

The concept of capacity is generalized for channels with feedback as follows. Let $\mathcal{D}_T = \{\{p(\alpha_t|\alpha^{t-1}, \beta^{t-1})\}_{t=1}^{T}\}$ be the set of all input distributions. For finite T, the capacity of a channel $\{p(\beta_t|\alpha^t, \beta^{t-1})\}_{t=1}^{T}$ is:

$$C_T = \sup_{\mathcal{D}_T} \frac{1}{T} I(A^T \rightarrow B^T) \tag{4}$$

4 Interactive Systems as Channels with Memory and Feedback

(General) Interactive Information Hiding Systems ([2]), are a variant of probabilistic automata in which we separate actions in secret and observable; "interactive" means that secret and observable actions can interleave and influence each other.

Definition 2. *A general* IIHS *is a quadruple* $\mathcal{I} = (M, \mathcal{A}, \mathcal{B}, \mathcal{L}_\tau)$*, where* M *is a probabilistic automaton* $(\mathcal{S}, \mathcal{L}, \hat{s}, \vartheta)$*,* $\mathcal{L} = \mathcal{A} \cup \mathcal{B} \cup \mathcal{L}_\tau$ *where* $\mathcal{A}$*,* $\mathcal{B}$*, and* $\mathcal{L}_\tau$ *are pairwise disjoint sets of secret, observable, and internal actions respectively, and* $\vartheta(s) \subseteq \mathcal{D}(\mathcal{B} \cup \mathcal{L}_\tau \times \mathcal{S})$ *implies* $|\vartheta(s)| \leq 1$*, for all* s*. The condition on* ϑ *ensures that all observable transitions are fully probabilistic.*

Assumption. In this paper we assume that general IIHSs are *normalized*, i.e. once unfolded, all the transitions between two consecutive levels have either secret labels only, or observable labels only. Moreover, the occurrences of secret and observable labels alternate between levels. We will call *secret states* the states from which only secrets-labeled transitions are possible, and *observable states* the others. Finally, we assume that for every s and ℓ there exists a unique r such that $s \xrightarrow{\ell} r$. Under this assumption we have that the traces of a computation determine the final state, as expressed by the next proposition. In the following $trace_{\mathcal{A}}$ and $trace_{\mathcal{B}}$ indicate the projection of the traces on secret and observable actions, respectively. Given a general IIHS, it is always possible to find an equivalent one that satisfies this assumptions. The interested reader can find in [1] the formal definition of the transformation.

Proposition 1. *Let* $\mathcal{I} = (M, \mathcal{A}, \mathcal{B}, \mathcal{L}_\tau)$ *be a general* IIHS. *Consider two paths* σ *and* σ'*. Then,* $trace_{\mathcal{A}}(\sigma) = trace_{\mathcal{A}}(\sigma')$ *and* $trace_{\mathcal{B}}(\sigma) = trace_{\mathcal{B}}(\sigma')$ *implies* $\sigma = \sigma'$.

In the following, we will consider two particular cases: the *fully probabilistic* IIHSs, where there is no nondeterminism, and the *secret -nondeterministic* IIHSs, where each secret choice is fully nondeterministic. The latter will be called simply IIHSs.

Definition 3. *Let* $\mathfrak{I} = ((\mathcal{S}, \mathcal{L}, \hat{s}, \vartheta), \mathcal{A}, \mathcal{B}, \mathcal{L}_\tau)$ *be a general* IIHS. *Then* $\mathfrak{I}$ *is:*

- *fully probabilistic if* $\vartheta(s) \subseteq \mathcal{D}(\mathcal{A} \times \mathcal{S})$ *implies* $|\vartheta(s)| \leq 1$ *for each* $s \in \mathcal{S}$.
- *secret-nondeterministic if* $\vartheta(s) \subseteq \mathcal{D}(\mathcal{A} \times \mathcal{S})$ *implies that for each* $s \in \mathcal{S}$ *there exist* s_i*' such that* $\vartheta(s) = \{\delta(a_i, s_i)\}_{i=1}^n$.

We show now how to construct a channel with memory and feedback from IIHSs. We will see that an IIHS corresponds precisely to a channel as determined by its stochastic kernel, while a fully probabilistic IIHS determines, additionally, the input distribution. In the following, we consider an IIHS $\mathfrak{I} = ((\mathcal{S}, \mathcal{L}, \hat{s}, \vartheta), \mathcal{A}, \mathcal{B}, \mathcal{L}_\tau)$ is in *normalized form*. Given a path σ of length $2t-1$, we denote $trace_{\mathcal{A}}(\sigma)$ by α^t, and $trace_{\mathcal{B}}(\sigma)$ by β^{t-1}.

Definition 4. *For each* t*, the channel's stochastic kernel corresponding to* $\mathfrak{I}$ *is defined as* $p(\beta_t|\alpha^t, \beta^{t-1}) = \vartheta(q)(\beta_t, q')$*, where* q *is the state reached from the root via the path* σ *whose input-trace is* α^t *and output trace* β^{t-1}.

Note that q and q' in previous definitions are well defined: by Proposition 1, q is unique, and since the choice of β_t is fully probabilistic, q' is also unique.

If $\mathfrak{I}$ is fully probabilistic, then it determines also the input distribution and the dependency of α_t upon β^{t-1} (feedback) and α^{t-1}.

Definition 5. *If* $\mathfrak{I}$ *is fully probabilistic, the associated channel has a conditional input distribution for each* t *defined as* $p(\alpha_t|\alpha^{t-1}, \beta^{t-1}) = \vartheta(q)(\alpha_t, q')$*, where* q *is the state reached from the root via the path* σ *whose input-trace is* α^{t-1} *and output trace is* β^{t-1}.

4.1 Lifting the Channel Inputs to Reaction Functions

Definitions 4 and 5 define the joint probabilities $p(\alpha^t, \beta^t)$ for a fully probabilistic IIHS. We still need to show in what sense these define a information-theoretic channel.

The $\{p(\beta_t|\alpha^t, \beta^{t-1})\}_{t=1}^T$ determined by the IIHS correspond to a channel's stochastic kernel. The problem resides in the conditional probability of $\{p(\alpha_t|\alpha^{t-1}, \beta^{t-1})\}_{t=1}^T$. In an information-theoretic channel, the value of α_t is determined in the encoder by a deterministic function $\varphi_t(\beta^{t-1})$. However, inside the encoder there is no possibility for a probabilistic description of α_t. Furthermore, in our setting the concept of encoder makes no sense as there is no information to encode. A solution to this problem is to externalize the probabilistic behavior of α_t: the code functions become simple *reaction functions* φ_t that depend only on β^{t-1} (the message w does not play a role any more), and these reaction functions are endowed with a probability distribution that generates the probabilistic behavior of the values of α_t.

Definition 6. *A* reactor *is a distribution on reaction functions, i.e., a stochastic kernel* $\{p(\varphi_t|\varphi^{t-1})\}_{t=1}^T$. *A reactor* R *is* consistent with a fully probabilistic IIHS $\mathcal{I}$ *if it induces the compatible distribution* $Q(\varphi^T, \alpha^T, \beta^T)$ *such that, for every* $1 \leq t \leq T$, $Q(\alpha_t|\alpha^{t-1}, \beta^{t-1}) = p(\alpha_t|\alpha^{t-1}, \beta^{t-1})$*, where the latter is the probability distribution induced by* $\mathfrak{I}$.

The main result of this section states that for any fully probabilistic IIHS there is a reactor that generates the probabilistic behavior of the IIHS.

Theorem 3. *Given a fully probabilistic* IIHS $\mathcal{I}$ *, we can construct a channel with memory and feedback, and probability distribution* $Q(\varphi^T, \alpha^T, \beta^T)$*, which corresponds to* $\mathcal{I}$ *in the sense that, for every* t*,* α^t *and* β^t*, with* $1 \leq t \leq T$*,* $Q(\alpha^t, \beta^t) \overset{\text{def}}{=} \sum_{\varphi^T} Q(\varphi^T, \alpha^t, \beta^t) = p(\alpha^t, \beta^t)$ *holds, where* $p(\alpha^t, \beta^t)$ *is the joint probability of input and output traces induced by* $\mathcal{I}$*.*

Corollary 1. *Let a* $\mathcal{I}$ *be a fully probabilistic* IIHS. *Let* $\{p(\beta_t|\alpha^t, \beta^{t-1})\}_{t=1}^T$ *be a sequence of stochastic kernels and* $\{p(\alpha_t|\alpha^{t-1}, \beta^{t-1})\}_{t=1}^T$ *a sequence of input distributions defined by* $\mathcal{I}$ *according to Definitions 4 and 5. Then the reactor* $R = \{p(\varphi_t|\varphi^{t-1})\}_{t=1}^T$ *compatible with respect to the* $\mathcal{I}$ *is given by:*

$$p(\varphi_1) = p(\alpha_1|\alpha^0, \beta^0) = p(\alpha_1) \tag{5}$$

$$p(\varphi_t|\varphi^{t-1}) = \prod_{\beta^{t-1}} p(\varphi_t(\beta^{t-1})|\varphi^{t-1}(\beta^{t-2}), \beta^{t-1}), \quad 2 \leq t \leq T \tag{6}$$

Figure 3 depicts the model for IIHS. Note that, in relation to Figure 2, there are some simplifications: (1) no message w is needed; (2) the decoder is not used. At the beginning, a reaction function sequence φ^T is chosen and then the channel is used T times. At each usage t, the encoder decides the next input symbol α_t based on the reaction function φ_t and the output fed back β^{t-1}. Then the channel produces an output β_t based on the stochastic kernel $p(\beta_t|\alpha^t, \beta^{t-1})$. The output is then fed back to the encoder with a delay one.

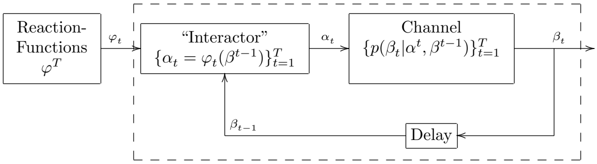


Fig. 3. Channel with memory and feedback model for IIHS

We conclude this section by remarking an intriguing coincidence: The notion of reaction function sequence φ^T, on the IIHSs, corresponds to the notion of deterministic scheduler. In fact, each reaction function φ_t selects the next step, α_t, on the basis of the β^{t-1} and α^{t-1} (generated by φ^{t-1}), and β^{t-1}, α^{t-1} represent the path until that state.

5 Leakage in Interactive Systems

In this section we propose a notion of information flow based on our model. We follow the idea of defining leakage and maximum leakage using the concepts of mutual information and capacity (see for instance [4]), making the necessary adaptations.

Since the directed information $I(A^T \rightarrow B^T)$ is a measure of how much information flows from A^T to B^T in a channel with feedback (cfr. Section 3.1), it is natural to consider it as a measure of leakage of information by the protocol.

Definition 7. *The information leakage of an* IIHS *is defined as:* $I(A^T \rightarrow B^T) = \sum_{t=1}^{T} H(A_t|A^{t-1}, B^{t-1}) - H(A^T|B^T)$.

Note that $\sum_{t=1}^{T} H(A_t|A^{t-1}, B^{t-1})$ can be seen as the entropy H_R of reactor R.

Compare this definition with the classical Information-theoretic approach to information leakage: when there is no feedback, the leakage is defined as:

$$I(A^T; B^T) = H(A^T) - H(A^T|B^T) \tag{7}$$

The principle behind (7) is that the leakage is equal to the difference between the *a priori uncertainty* $H(A^T)$ and the *a posteriori uncertainty* $H(A^T|B^T)$ (gain in knowledge about the secret by observing the output). Our definition maintains the same principle, with the proviso that the a priori uncertainty is now represented by H_R.

5.1 Maximum Leakage as Capacity

In the case of secret-nondeterministic IIHS, we have a stochastic kernel but no distribution on the code functions. In this case it seems natural to consider the worst leakage over all possible distributions on code functions. This is exactly the concept of capacity.

Definition 8. *The* maximum leakage *of an* IIHS *is defined as the capacity* C_T *of the associated channel with memory and feedback.*

6 Modeling IIHSs as Channels: An Example

In this section we show the application of our approach to the *Cocaine Auction Protocol* [17]. Let us imagine a situation where several mob individuals are gathered around a table. An auction is about to be held in which one of them offers his next shipment of cocaine to the highest bidder. The seller describes the merchandise and proposes a starting price. The others then bid increasing amounts until there are no bids for 30 consecutive seconds. At that point the seller declares the auction closed and arranges a secret appointment with the winner to deliver the goods.

The basic protocol is fairly simple and is organized as a succession of rounds of bidding. Round i starts with the seller announcing the bid price b_i for that round. Buyers have t seconds to make an offer (i.e. to say yes, meaning "I'm willing to buy at the current bid price b_i"). As soon as one buyer anonymously says yes, he becomes the winner w_i of that round and a new round begins. If nobody says anything for t seconds, round i is concluded by timeout and the auction is won by the winner w_{i-1} of the previous round, if one exists. If the timeout occurs during round 0, this means that nobody made any offers at the initial price b_0, so there is no sale.

Although our framework allows the forrmalization of this protocol for an arbitrary number of bidders and bidding rounds, for illustration purposes, we will consider the case of two bidders (*Candlemaker* and *Scarface*) and two rounds of bids. Furthermore, we assume that the initial bid is always 1 dollar, so the first bid does not need to be announced by the seller. In each turn the seller can choose how much he wants to increase the actual bid. This is done by adding an increment to the last bid. There are two options of increments, namely inc_1 (1 dollar) and inc_2 (2 dollars). In that way, b_{i+1} is either $b_i + inc_1$ or $b_i + inc_2$. We can describe this protocol as a *normalized* IIHS $\mathcal{I} = (M, \mathcal{A}, \mathcal{B}, \mathcal{L}_\tau)$, where $\mathcal{A} = \{$Candlemaker, Scarface, $a^*\}$ is the set of secret actions, $\mathcal{B} = \{inc_1, inc_2, b_*\}$ is the set of observable actions, $\mathcal{L}_\tau = \emptyset$ is the set of hidden actions, and the probabilistic automaton M is represented in Figure 4. For clarity reasons, we omit transitions with probability 0 in the automaton. Note that the special secret action a_* represents the situation where neither *Candlemaker* nor *Scarface* bid. The special observable action b_* is only possible after no one has bidden, and signalizes the end of the auction and, therefore, no bid is allowed anymore.

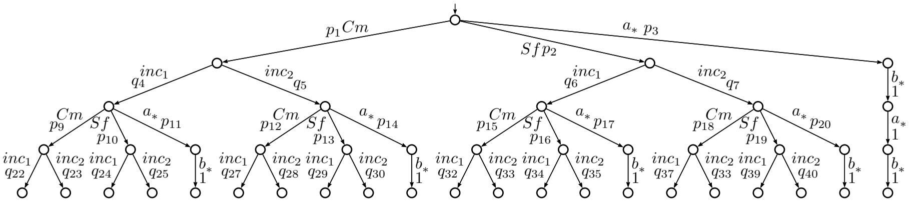


Fig. 4. Cocaine Auction example

Table 4 shows all the stochastic kernels for this example. The formalization of this protocol in terms of IIHSs using our framework makes it possible to prove the claim in[17] suggesting that if the seller knows the identity of the bidders then the (strong) anonymity guaranties are not provided anymore.

Table 4. Stochastic kernels for the Cocaine Auction example

$\alpha_1 \to \beta_1$	inc_1	inc_2	b_*
Candlemaker	q_4	q_5	0
Scarface	q_6	q_7	0
a^*	0	0	1

(a) $t=1, p(\beta_1|\alpha^1, \beta^0)$

$\alpha_1, \beta_1, \alpha_2 \to \beta_2$	Cheap	Expensive	b_*
Candlemaker,inc_1,*Candlemaker*	q_{22}	q_{23}	0
Candlemaker,inc_1,*Scarface*	q_{24}	q_{25}	0
Candlemaker,inc_1,a_*	0	0	1
Candlemaker,inc_2,*Candlemaker*	q_{27}	q_{28}	0
Candlemaker,inc_2,*Scarface*	q_{29}	q_{30}	0
Candlemaker,inc_2,a_*	0	0	1
Scarface,inc_1,*Candlemaker*	q_{32}	q_{33}	0
Scarface,inc_1,*Scarface*	q_{34}	q_{35}	0
Scarface,inc_1,a_*	0	0	1
Scarface,inc_2,*Candlemaker*	q_{37}	q_{38}	0
Scarface,inc_2,*Scarface*	q_{39}	q_{40}	0
Scarface,inc_2,a_*	0	0	1
a_*,b_*,a_*	0	0	1
All other lines	0	0	1

(b) $t=2, p(\beta_2|\alpha^2, \beta^1)$

7 Topological Properties of IIHSs and Their Capacity

In this section we show how to extend to IIHSs the notion of pseudometric defined in [8] for Concurrent Labelled Markov Chains, and we prove that the capacity of the corresponding channels is a continuous function on this pseudometric. The metric construction is sound for general IIHSs, but the result on capacity is only valid for secret-nondeterministic IIHSs.

Given a set of states S, a pseudometric (or distance) is a function d that yields a non-negative real number for each pair of states and satisfies the following: $d(s,s) = 0$; $d(s,t) = d(t,s)$, and $d(s,t) \leq d(s,u) + d(u,t)$. We say that a pseudometric d is c-bounded if $\forall s,t : d(s,t) \leq c$, where c is a positive real number. We now define a complete lattice on pseudometrics, and define the distance between IIHSs as the greatest fixpoint of a distance transformation, in line with the coinductive theory of bisimilarity.

Definition 9. *$\mathcal{M}$ is the class of 1-bounded pseudometrics on states with the ordering $d \preceq d'$ if $\forall s,s' \in S : d(s,s') \geq d'(s,s')$.*

It is easy to see that $(\mathcal{M}, \preceq)$ is a complete lattice. In order to define pseudometrics on IIHSs, we now need to lift the pseudometrics on states to pseudometrics on distributions in $\mathcal{D}(\mathcal{L} \times S)$. Following standard lines [20,8,7], we apply the construction based on the Kantorovich metric [11].

Definition 10. *For $d \in \mathcal{M}$, and $\mu, \mu' \in \mathcal{D}(\mathcal{L} \times S)$, we define $d(\mu,\mu')$ (overloading the notation d) as $d(\mu,\mu') = \max \sum_{(\ell_i,s_i)\in\mathcal{L}\times S}(\mu(\ell_i,s_i) - \mu'(\ell_i,s_i))x_i$ where the maximization is on all possible values of the x_i's, subject to the constraints $0 \leq x_i \leq 1$ and $x_i - x_j \leq \hat{d}((\ell_i,s_i),(\ell_j,s_j))$, where $\hat{d}((\ell_i,s_i),(\ell_j,s_j)) = 1$ if $\ell_i \neq \ell_j$, and $\hat{d}((\ell_i,s_i),(\ell_j,s_j)) = d(s_i,s_j)$ otherwise.*

It can be shown that with this definition m is a pseudometric on $\mathcal{D}(\mathcal{L} \times S)$.

Definition 11. *$d \in \mathcal{M}$ is a* bisimulation metric *if, for all $\epsilon \in [0,1)$, $d(s,s') \leq \epsilon$ implies that if $s \longrightarrow \mu$, then there exists some μ' such that $s' \longrightarrow \mu'$ and $d(\mu,\mu') \leq \epsilon$.*

The greatest bisimulation metric is $d_{max} = \bigsqcup\{d \in \mathcal{M} \mid d \text{ is a bisimulation metric}\}$. We now characterize d_{max} as a fixed point of a monotonic function Φ on $\mathcal{M}$. For simplicity, from now on we consider only the distance between states belonging to different IIHSs with disjoint sets of states.

Definition 12. *Given two* IIHSs *with transition relations θ and θ' respectively, and a preudometric d on states, define $\Phi : \mathcal{M} \to \mathcal{M}$ as:*

$$\Phi(d)(s,s') = \begin{cases} \max_i d(s_i,s'_i) & \textit{if} \quad \vartheta(s) = \{\delta_{(a_1,s_1)},\ldots,\delta_{(a_m,s_m)}\} \\ & \textit{and} \;\; \vartheta'(s') = \{\delta_{(a_1,s'_1)},\ldots,\delta_{(a_m,s'_m)}\} \\ d(\mu,\mu') & \textit{if}\; \vartheta(s) = \{\mu\} \;\textit{and}\; \vartheta'(s') = \{\mu'\} \\ 0 & \textit{if}\; \vartheta(s) = \vartheta'(s') = \emptyset \\ 1 & \textit{otherwise} \end{cases}$$

It is easy to see that the definition of Φ is a particular case of the function F defined in [8,7]. Hence it can be proved, by adapting the proofs of the analogous results in [8,7], that $F(d)$ is a pseudometric, and that d is a bisimulation metric iff $d \preceq \Phi(d)$. This implies that $d_{max} = \bigsqcup\{d \in \mathcal{M} \mid d \preceq \Phi(d)\}$, and still as a particular case of F in [8,7], we have that Φ is monotonic on $\mathcal{M}$. By Tarski's fixed point theorem, d_{max} is the greatest fixed point of Φ. Furthermore, in [1] we show that d_{max} is indeed a bisimulation metric, and that it is the greatest bisimulation metric. In addition, the finite branchingness of IIHSs ensures that the closure ordinal of Φ is ω (cf. Lemma 3.10 in the full version of [8]). Therefore one can show that $d_{max} = \bigsqcap\{\Phi^i(\top) \mid i \in \mathbb{N}\}$, where $\top$ is the greatest pseudometric (i.e. $\top(s, s') = 0$ for every s, s'), and $\Phi^0(\top) = \top$.

Given two IIHSs $\mathcal{I}$ and $\mathcal{I}'$, with initial states s and s' respectively, we define the distance between $\mathcal{I}$ and $\mathcal{I}'$ as $d(\mathcal{I}, \mathcal{I}') = d_{max}(s, s')$. Next theorem states the continuity of the capacity w.r.t. the metric on IIHSs. It is crucial that they are secret-nondeterministic (while the definition of the metric holds in general).

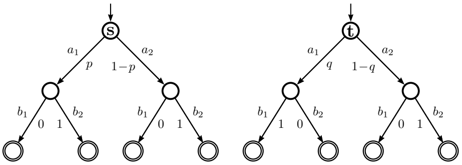
Theorem 4. *Consider two normalized* IIHSs $\mathcal{I}$ *and* $\mathcal{I}'$, *and fix a* $T > 0$. *For every* $\epsilon > 0$ *there exists* $\nu > 0$ *such that if* $d(\mathcal{I}, \mathcal{I}') < \nu$ *then* $|C_T(\mathcal{I}) - C_T(\mathcal{I}')| < \epsilon$.

We conclude this section with an example showing that the continuity result for the capacity does not hold if the construction of the channel is done starting from a system in which the secrets are endowed with a probability distribution. This is also the reason why we could not simply adopt the proof technique of the continuity result in [8] and we had to come up with a different reasoning.

Example 3. Consider the two following programs, where a_1, a_2 are secrets, b_1, b_2 are observable, $\|$ is the parallel operator, and $+_p$ is a binary probabilistic choice that assigns probability p to the left branch, and probability $1 - p$ to the right one.

s) $(\mathit{send}(a_1) +_p\ \mathit{send}(a_2)) \parallel \mathit{receive}(x).\mathit{output}(b_2)$
t) $(\mathit{send}(a_1) +_q\ \mathit{send}(a_2)) \parallel \mathit{receive}(x).\mathit{if}\ x = a_1\ \mathit{then}\ \mathit{output}(b_1)\ \mathit{else}\ \mathit{output}(b_2)$.

Table 5 shows the fully probabilistic IIHSs corresponding to these programs, and their associated channels, which in this case (since the secret actions are all at the top-level) are classic channels, i.e. memoryless and without feedback. As usual for classic channels, they do not depend on p and q. It is easy to see that the capacity of the first channel is 0 and the capacity of the second one is 1. Hence their difference is 1, independently from p and q.



s	b_1	b_2
a_1	0	1
a_2	0	1

(a)

t	b_1	b_2
a_1	1	0
a_2	0	1

(b)

Table 5. The IIHSs of Example 3 and their corresponding channels

Let now $p = 0$ and $q = \epsilon$. It is easy to see that the distance between s and t is ϵ. Therefore (when the automata have probabilities on the secrets), the capacity is not a continuous function of the distance.

8 Conclusion and Future Work

In this paper we have investigated the problem of information leakage in interactive systems, and we have proved that these systems can be modeled as channels with memory and feedback. The situation is summarized in Table 6(a). The comparison with the classical situation of non-interactive systems is represented in (b). Furthermore, we have proved that the channel capacity is a continuous function of the kantorovich metric.

IIHSs **as automata**	IIHSs **as channels**	**Notion of leakage**
Normalized IIHSs with nondeterministic inputs and probabilistic outputs	Sequence of stochastic kernels $\{p(\beta_t\|\alpha^t, \beta^{t-1})\}_{t=1}^T$	Leakage as capacity
Normalized IIHSs with a deterministic scheduler solving the nondeterminism	Sequence of stochastic kernels $\{p(\beta_t\|\alpha^t, \beta^{t-1})\}_{t=1}^T$ + reaction function seq. φ^T	
Fully probabilistic normalized IIHSs	Sequence of stochastic kernels $\{p(\beta_t\|\alpha^t, \beta^{t-1})\}_{t=1}^T$ + reactor $\{p(\varphi_t\|\varphi^{t-1})\}_{t=1}^T$	Leakage as directed information $I(A^T \to B^T)$

(a)

Classical channels	**Channels with memory and feedback**
The protocol is modeled in independent uses of the channel, often a unique use.	The protocol is modeled in several consecutive uses of the channel.
The channel is from $\mathcal{A}^T \to \mathcal{B}^T$, i.e., its input is a single string $\alpha^T = \alpha_1 \dots \alpha_T$ of secret symbols and its output is a single string $\beta^T = \beta_1 \dots \beta_T$ of observable symbols.	The channel is from $\mathcal{F} \to \mathcal{B}$, i.e. its input is a reaction function φ_t and its output is an observable β_t.
The channel is memoryless and in general implicitly it is assumed the absence of feedback.	The channel has memory. Despite the fact that the channel from $\mathcal{F} \to \mathcal{B}$ does not have feedback, the internal stochastic kernels do.
The capacity is calculated using information $I(A^T; B^T)$.	The capacity is calculated using mutual directed information $I(A^T \to B^T)$.

(b)

Table 6.

For future work we would like to provide algorithms to compute the leakage and maximum leakage of interactive systems. These problems result very challenging given the exponential growth of reaction functions (needed to compute the leakage) and the quantification over infinitely many reactors (given by the definition of maximum leakage in terms of capacity). One possible solution is to study the relation between deterministic schedulers and sequence of reaction functions. In particular, we believe that for each sequence of reaction functions and distribution over it there exists a probabilistic scheduler for the automata representation of the secret-nondeterministic IIHS.

In this way, the problem of computing the leakage and maximum leakage would reduce to a standard probabilistic model checking problem (where the challenge is to compute probabilities ranging over infinitely many schedulers).

In addition, we plan to investigate measures of leakage for interactive systems other than mutual information and capacity.

References

1. Alvim, M.S., Andrés, M.E., Palamidessi, C.: Information Flow in Interactive Systems (2010), http://hal.archives-ouvertes.fr/inria-00479672/en/
2. Andrés, M.E., Palamidessi, C., van Rossum, P., Smith, G.: Computing the leakage of information-hiding systems. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 373–389. Springer, Heidelberg (2010)
3. Bohannon, A., Pierce, B.C., Sjöberg, V., Weirich, S., Zdancewic, S.: Reactive noninterference. In: Proc. of CCS, pp. 79–90. ACM, New York (2009)
4. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Anonymity protocols as noisy channels. Inf. and Comp. 206(2-4), 378–401 (2008)
5. Clark, D., Hunt, S., Malacaria, P.: Quantified interference for a while language. In: Proc. of QAPL 2004. ENTCS, vol. 112, pp. 149–166. Elsevier, Amsterdam (2005)
6. Cover, T.M., Thomas, J.A.: Elements of Information Theory. J. Wiley & Sons, Inc., Chichester (1991)
7. Deng, Y., Chothia, T., Palamidessi, C., Pang, J.: Metrics for action-labelled quantitative transition systems. In: Proc. of QAPL. ENTCS, vol. 153, pp. 79–96. Elsevier, Amsterdam (2006)
8. Desharnais, J., Jagadeesan, R., Gupta, V., Panangaden, P.: The metric analogue of weak bisimulation for probabilistic processes. In: Proc. of LICS, pp. 413–422. IEEE, Los Alamitos (2002)
9. Ebay website, http://www.ebay.com/
10. Ebid website, http://www.ebid.net/
11. Kantorovich, L.: On the transfer of masses (in Russian). Doklady Akademii Nauk 5(1), 1–4 (1942); Translated in Management Science 5(1), 1–4 (1958)
12. Malacaria, P.: Assessing security threats of looping constructs. In: Proc. of POPL, pp. 225–235. ACM, New York (2007)
13. Massey, J.L.: Causality, feedback and directed information. In: Proc. of the 1990 Intl. Symposium on Information Theory and its Applications (1990)
14. Mercadolibre website, http://www.mercadolibre.com/
15. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis. Tech. Rep. MIT/LCS/TR-676 (1995)
16. Smith, G.: On the foundations of quantitative information flow. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 288–302. Springer, Heidelberg (2009)
17. Stajano, F., Anderson, R.J.: The cocaine auction protocol: On the power of anonymous broadcast. In: Information Hiding, pp. 434–447 (1999)
18. Subramanian, S.: Design and verification of a secure electronic auction protocol. In: Proc. of SRDS, pp. 204–210. IEEE, Los Alamitos (1998)
19. Tatikonda, S., Mitter, S.K.: The capacity of channels with feedback. IEEE Transactions on Information Theory 55(1), 323–349 (2009)
20. van Breugel, F., Worrell, J.: Towards quantitative verification of probabilistic transition systems. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 421–432. Springer, Heidelberg (2001)
21. Vickrey, W.: Counterspeculation, Auctions, and Competitive Sealed Tenders. The Journal of Finance 16(1), 8–37 (1961)

From Multi to Single Stack Automata

Mohamed Faouzi Atig

LIAFA, CNRS & Univ. of Paris 7, Case 7014, 75205 Paris 13, France
atig@liafa.jussieu.fr

Abstract. We investigate the issue of reducing the verification problem of multi-stack machines to the one for single-stack machines. For instance, elegant (and practically efficient) algorithms for bounded-context switch analysis of multi-pushdown systems have been recently defined based on reductions to the reachability problem of (single-stack) pushdown systems [10,18]. In this paper, we extend this view to both bounded-phase visibly pushdown automata (BVMPA) [16] and ordered multi-pushdown automata (OMPA) [1] by showing that each of their emptiness problem can be reduced to the one for a class of single-stack machines. For these reductions, we introduce *effective generalized pushdown automata* (EGPA) where operations on stacks are (1) pop the top symbol of the stack, and (2) push a word in some (effectively) given set of words L over the stack alphabet, assuming that L is in some class of languages for which checking whether L intersects regular languages is decidable. We show that the automata-based saturation procedure for computing the set of predecessors in standard pushdown automata can be extended to prove that for EGPA too the set of all predecessors of a regular set of configurations is an effectively constructible regular set. Our reductions from OMPA and BVMPA to EGPA, together with the reachability analysis procedure for EGPA, allow to provide conceptually simple algorithms for checking the emptiness problem for each of these models, and to significantly simplify the proofs for their 2ETIME upper bounds (matching their lower-bounds).

1 Introduction

In the last few years, a lot of effort has been devoted to the verification problem for models of concurrent programs (see, e.g., [5,3,16,9,2]). Pushdown automata have been proposed as an adequate formalism to describe sequential programs with procedure calls [8,14]. Therefore, it is natural to model recursive concurrent programs as multi-stack automata. In general, multi-stack automata are Turing powerful and hence come along with undecidability of basic decision problems [13]. To overcome this barrier, several subclasses of pushdown automata with multiple stacks have been proposed and studied in the literature.

Context-bounding has been proposed in [12] as a suitable technique for the analysis of multi-stack automata. The idea is to consider only runs of the automaton that can be divided into a given number of contexts, where in each context pop and push operations are exclusive to one stack. Although the state space which may be explored is still unbounded in presence of recursive procedure calls, the context-bounded reachability

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 117–131, 2010.

problem is NP-complete even in this case [11]. In fact, context-bounding provides a very useful tradeoff between computational complexity and verification coverage.

In [16], La Torre et al. propose a more general definition of the notion of a context. For that, they define the class of *bounded-phase visibly multi-stack pushdown automata* (BVMPA) where only those runs are taken into consideration that can be split into a given number of phases, where each phase admits pop operations of one particular stack only. In the above case, the emptiness problem is decidable in double exponential time by reducing it to the emptiness problem for tree automata.

Another way to regain decidability is to impose some order on stack operations. In [6], Breveglieri et al. define *ordered multi-pushdown automata* (OMPA), which impose a linear ordering on stacks. Stack operations are constrained in such a way that a pop operation is reserved to the first non-empty stack. In [1], we show that the emptiness problem for OMPA is 2ETIME-complete[1]. The proof of this result lies in a complex encoding of OMPA into some class of grammars for which the emptiness problem is decidable.

In this paper, we investigate the issue of reducing the verification problem of multi-stack machines to the one for single-stack machines. We believe that this is a general paradigm for understanding the expressive power and for establishing decidability results for various classes of concurrent program models. For instance, elegant (and practically efficient) algorithms for bounded-context switch analysis of multi-pushdown systems have been recently defined based on reductions to the reachability problem of (single-stack) pushdown systems [10,18]. We extend this view to both OMPA and BVMPA by showing that each of their emptiness problem can be reduced to the one for a class of single-stack machines. For these reductions, we introduce *effective generalized pushdown automata* (EGPA) where operations on stacks are (1) pop the top symbol of the stack, and (2) push a word in some (effectively) given set of words L over the stack alphabet, assuming that L is in some class of languages for which checking whether L intersects a given regular language is decidable. Observe that L can be any finite union of languages defined by a class of automata closed under intersection with regular languages and for which the emptiness problem is decidable (e.g., pushdown automata, Petri nets, lossy channel machines, etc). Then, we show that the automata-based saturation procedure for computing the set of predecessors in standard pushdown automata [4] can be extended to prove that for EGPA too the set of all predecessors of a regular set of configurations is a regular set and effectively constructible. As an immediate consequence of this result, we obtain similar decidability results of the decision problems for EGPA like the ones obtained for pushdown automata.

Then, we show that, given an OMPA $\mathcal{M}$ with n stacks, it is possible to construct an EGPA $\mathcal{P}$, whose pushed languages are defined by OMPA with $(n-1)$ stacks, such that the emptiness problem for $\mathcal{M}$ is reducible to its corresponding problem for $\mathcal{P}$. The EGPA $\mathcal{P}$ is constructed such that the following invariant is preserved: The state and the content of the stack of $\mathcal{P}$ are the same as the state and the content of the n-th stack of $\mathcal{M}$ when its first $(n-1)$ stacks are empty. Then, we use the saturation procedure for

[1] Recall that 2ETIME is the class of all decision problems solvable by a deterministic Turing machine in time $2^{2^{dn}}$ for some constant d.

EGPA to show, by induction on the number of stacks n, that the emptiness problem of an OMPA is in 2ETIME with respect to n (matching its lower-bound [1]).

Another application of EGPA is to show that, given a k-phase BVMPA $\mathcal{M}$ with n stacks, it is possible to construct an EGPA $\mathcal{P}$, whose pushed languages are defined by $(k-1)$-phase BVMPA with n stacks, such that the emptiness problem of $\mathcal{M}$ can be reduced to compute the set of predecessors of a regular set of configurations of $\mathcal{P}$. Then, we exploit the saturation procedure for EGPA to show, by induction on the number of phases k, that the emptiness problem for a BVMPA is in 2ETIME with respect to k (matching its lower-bound [17]).

Related works: To the best of our knowledge, the class of effective generalized pushdown automata that we define in this paper is the first non-trivial extension of pushdown automata that allows to push a non-regular language (e.g., Petri nets languages) into the stack which is not the case of prefix-recognizable graphs [7]. Moreover, our reductions from OMPA and BVMPA to EGPA, together with the reachability analysis procedure for EGPA, provide conceptually simple algorithms for checking the emptiness problem for each of these models, and proving their 2ETIME upper bounds.

2 Preliminaries

In this section, we introduce some basic definitions and notations that will be used in the rest of the paper.

Integers: Let $\mathbb{N}$ be the set of natural numbers. For every $i, j \in \mathbb{N}$ such that $i \leq j$, we use $[i, j]$ (resp. $[i, j[$) to denote the set $\{k \in \mathbb{N} \mid i \leq k \leq j\}$ (resp. $\{k \in \mathbb{N} \mid i \leq k < j\}$).

Words and languages: Let Σ be a finite alphabet. We denote by Σ^* (resp. Σ^+) the set of all words (resp. non empty words) over Σ, and by ϵ the empty word. A language is a (possibly infinite) set of words. We use Σ_ϵ and $Lang(\Sigma)$ to denote respectively the set $\Sigma \cup \{\epsilon\}$ and the set of all languages over Σ. Let u be a word over Σ. The length of u is denoted by $|u|$. For every $j \in [1, |u|]$, we use $u(j)$ to denote the j^{th} letter of u. We denote by u^R the mirror of u.

Let Θ be a subset of Σ. Given a word $v \in \Sigma^*$, we denote by $v|_\Theta$ the projection of v over Θ, i.e., the word obtained from v by erasing all the symbols that are not in Θ. This definition is extended to languages as follows: If L is a language over Σ, then $L|_\Theta = \{v|_\Theta \mid v \in L\}$.

Transition systems: A transition system is a triplet $\mathcal{T} = (C, \Sigma, \rightarrow)$ where: (1) C is a (possibly infinite) set of configurations, (2) Σ is a finite set of labels (or actions) such that $C \cap \Sigma = \emptyset$, and (3) $\rightarrow \subseteq C \times \Sigma_\epsilon \times C$ is a transition relation. We write $c \xrightarrow{a}_{\mathcal{T}} c'$ whenever c and c' are two configurations and a is an action such that $(c, a, c') \in \rightarrow$.

Given two configurations $c, c' \in C$, a finite run ρ of $\mathcal{T}$ from c to c' is a finite sequence $c_0 a_1 c_1 \cdots a_n c_n$, for some $n \geq 1$, such that: (1) $c_0 = c$ and $c_n = c'$, and (2) $c_i \xrightarrow{a_{i+1}}_{\mathcal{T}} c_{i+1}$ for all $i \in [0, n[$. In this case, we say that ρ has length n and is labelled by the word $a_1 a_2 \cdots a_n$.

Let $c, c' \in C$ and $u \in \Sigma^*$. We write $c \xRightarrow[n]{u}_{\mathcal{T}} c'$ if one of the following two cases holds: (1) $n = 0$, $c = c'$, and $u = \epsilon$, and (2) there is a run ρ of length n from c to

c' labelled by u. We also write $c \xRightarrow{u}{}^{*}_{\mathcal{T}} c'$ (resp. $c \xRightarrow{u}{}^{+}_{\mathcal{T}} c'$) to denote that $c \xRightarrow[n]{u}_{\mathcal{T}} c'$ for some $n \geq 0$ (resp. $n > 0$).

For every $C_1, C_2 \subseteq C$, let $\mathit{Traces}_{\mathcal{T}}(C_1, C_2) = \{u \in \Sigma^* \mid \exists (c_1, c_2) \in C_1 \times C_2\,,\ c_1 \xRightarrow{u}{}^{*}_{\mathcal{T}} c_2\}$ be the set of sequences of actions generated by the runs of $\mathcal{T}$ from a configuration in C_1 to a configuration in C_2, and let $\mathit{Pre}^*_{\mathcal{T}}(C_1) = \{c \in C \mid \exists (c', u) \in C_1 \times \Sigma^*\,,\ c \xRightarrow{u}{}^{*}_{\mathcal{T}} c'\}$ be the set of predecessor configurations of C_1.

Finite state automata: A finite state automaton (FSA) is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ where: (1) Q is the finite non-empty set of states, (2) Σ is the finite input alphabet, (3) $\Delta \subseteq (Q \times \Sigma_\epsilon \times Q)$ is the transition relation, (4) $I \subseteq Q$ is the set of initial states, and (5) $F \subseteq Q$ is the set of final states. We represent a transition (q, a, q') in Δ by $q \xrightarrow{a}_{\mathcal{A}} q'$. Moreover, if I' and F' are two subsets of Q, then we use $\mathcal{A}(I', F')$ to denote the finite state automaton defined by the tuple $(Q, \Sigma, \Delta, I', F')$.

The size of $\mathcal{A}$ is defined by $|\mathcal{A}| = (|Q| + |\Sigma|)$. We use $\mathcal{T}(\mathcal{A}) = (Q, \Sigma, \Delta)$ to denote the transition system associated with $\mathcal{A}$. The language accepted (or recognized) by $\mathcal{A}$ is given by $L(\mathcal{A}) = \mathit{Traces}_{\mathcal{T}(\mathcal{A})}(I, F)$.

3 Generalized Pushdown Automata

In this section, we introduce the class of generalized pushdown automata where operations on stacks are (1) pop the top symbol of the stack, and (2) push a word in some (effectively) given set of words L over the stack alphabet. A transition t is of the form $\delta(p, \gamma, a, p') = L$ where L is a (possibly infinite) set of words. Being in a configuration (q, w) where q is a state and w is a stack content, t can be applied if both $p = q$ and the content of the stack is of the form $\gamma w'$ for some w'. Taking the transition and reading the input letter a (which may be the empty word), the system moves to the successor configuration (p', uw') where $u \in L$ (i.e., the new state is p', and γ is replaced with a word u belonging to the language L). Formally, we have:

Definition 1 (Generalized pushdown automata). *A generalized pushdown automaton (GPA for short) is a tuple $\mathcal{P} = (P, \Sigma, \Gamma, \delta, p_0, \gamma_0, F)$ where:* (1) *P is the finite non-empty set of states,* (2) *Σ is the input alphabet,* (3) *Γ is the stack alphabet,* (4) *$\delta : P \times \Gamma \times \Sigma_\epsilon \times P \rightarrow \mathit{Lang}(\Gamma)$ is the transition function,* (5) *$p_0 \in P$ is the initial state,* (6) *$\gamma_0 \in \Gamma$ is the initial stack symbol, and* (7) *$F \subseteq P$ is the set of final states.*

Definition 2 (Effectiveness Property). *A GPA $\mathcal{P} = (P, \Sigma, \Gamma, \delta, p_0, \gamma_0, F)$ is effective if and only if for every finite state automaton $\mathcal{A}$ over the alphabet Γ, it is decidable whether $L(\mathcal{A}) \cap \delta(p, \gamma, a, p') \neq \emptyset$ for all $p, p' \in P$, $\gamma \in \Gamma$, and $a \in \Sigma_\epsilon$.*

A configuration of a GPA $\mathcal{P} = (P, \Sigma, \Gamma, \delta, p_0, \gamma_0, F)$ is a pair (p, w) where $p \in P$ and $w \in \Gamma^*$. The set of all configurations of $\mathcal{P}$ is denoted by $\mathit{Conf}(\mathcal{P})$. Similarly to the case of pushdown automata [4], we use the class of $\mathcal{P}$-automata as finite symbolic representation of a set of configurations of GPA. Formally, a $\mathcal{P}$-automaton is a FSA $\mathcal{A} = (Q_{\mathcal{A}}, \Gamma, \Delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}})$ such that $I_{\mathcal{A}} = P$. We say that a configuration (p, w) of $\mathcal{P}$ is accepted (or recognized) by $\mathcal{A}$ if $w \in L(\mathcal{A}(\{p\}, F_{\mathcal{A}}))$. The set of all configurations recognized by $\mathcal{A}$ is denoted by $L_{\mathcal{P}}(\mathcal{A})$. A set of configurations of $\mathcal{P}$ is said to be recognizable if and only if it is accepted by some $\mathcal{P}$-automaton.

The transition system $\mathcal{T}(\mathcal{P})$ associated with the generalized pushdown automaton $\mathcal{P}$ is defined by the tuple $(\mathit{Conf}(\mathcal{P}), \Sigma, \rightarrow)$ where $\rightarrow$ is the smallest transition relation such that: For every $p, p' \in P$, $\gamma \in \Gamma$, and $a \in \Sigma_\epsilon$, if $\delta(p, \gamma, a, p') \neq \emptyset$, then $(p, \gamma w) \xrightarrow{a}_{\mathcal{T}(\mathcal{P})} (p', uw)$ for all $u \in \delta(p, \gamma, a, p')$ and $w \in \Gamma^*$. Let $L(\mathcal{P}) = \mathit{Traces}_{\mathcal{T}(\mathcal{P})}(\{(p_0, \gamma_0)\}, F \times \{\epsilon\})$ denote the language accepted by $\mathcal{P}$.

Observe that pushdown automata can be seen as a particular class of effective GPA where $\delta(p, \gamma, a, p')$ is a finite set of words for all (p, γ, a, p').

On the other hand, we can show that the class of effective GPA is closed under concatenation, union, Kleene star, projection, homomorphism, and intersection with a regular language. However, effective GPA are not closed under intersection.

4 Computing the Set of Predecessors for a GPA

In this section, we show that the set of predecessors of a recognizable set of configurations of an effective GPA is recognizable and effectively constructible. This is done by adapting the construction given in [4]. On the other hand, it is easy to observe that the set of successors of a recognizable set of configurations of an effective GPA is not recognizable in general.

Theorem 1. *For every effective generalized pushdown automaton $\mathcal{P}$, and every $\mathcal{P}$-automaton $\mathcal{A}$, it is possible to construct a $\mathcal{P}$-automaton recognizing $Pre^*_{\mathcal{T}(\mathcal{P})}(L_{\mathcal{P}}(\mathcal{A}))$.*

Proof. Let $\mathcal{P} = (P, \Sigma, \Gamma, \delta, p_0, \gamma_0, F)$ be an effective generalized pushdown automata and $\mathcal{A} = (Q_{\mathcal{A}}, \Gamma, \Delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}})$ be a $\mathcal{P}$-automaton. Without loss of generality, we assume that $\mathcal{A}$ has no transition leading to an initial state. We compute $Pre^*_{\mathcal{T}(\mathcal{P})}(L_{\mathcal{P}}(\mathcal{A}))$ as the set of configurations recognized by a $\mathcal{P}$-automaton $\mathcal{A}_{pre^*} = (Q_{\mathcal{A}}, \Gamma, \Delta_{pre^*}, I_{\mathcal{A}}, F_{\mathcal{A}})$ obtained from $\mathcal{A}$ by means of a saturation procedure. Initially, we have $\mathcal{A}_{pre^*} = \mathcal{A}$. Then, the procedure adds new transitions to $\mathcal{A}_{pre^*}$, but no new states. New transitions are added according to the following saturation rule:

For every $p, p' \in P$, $\gamma \in \Gamma$, and $a \in \Sigma_\epsilon$, if $\delta(p, \gamma, a, p') \neq \emptyset$, then for every $q \in Q_{\mathcal{A}}$ such that $\delta(p, \gamma, a, p') \cap L(\mathcal{A}_{pre^*}(\{p'\}, \{q\})) \neq \emptyset$, add the transition (p, γ, q) to $\mathcal{A}_{pre^*}$

It is easy to see that the saturation procedure eventually reaches a fixed point because the number of possible new transitions is finite. Moreover, the saturation procedure is well defined since the emptiness problem of the language $\big(\delta(p, \gamma, a, p') \cap L(\mathcal{A}_{pre^*}(\{p'\}, \{q\}))\big)$ is decidable ($\mathcal{P}$ is an effective GPA). Then, the relation between the set of configurations recognized by $\mathcal{A}_{pre^*}$ and the set $Pre^*_{\mathcal{T}(\mathcal{P})}(L_{\mathcal{P}}(\mathcal{A}))$ is established by Lemma 1.

Lemma 1. $L_{\mathcal{P}}(\mathcal{A}_{pre^*}) = Pre^*_{\mathcal{T}(\mathcal{P})}(L_{\mathcal{P}}(\mathcal{A}))$. □

As an immediate consequence of Theorem 1, we obtain the decidability of the emptiness problem and the membership for effective generalized pushdown automata.

Theorem 2 (EMPTINESS, MEMBERSHIP). *The emptiness and the membership problems are decidable for effective generalized pushdown automata.*

5 Ordered Multi-Pushdown Automata

In this section, we first recall the definition of *multi-pushdown automata*. Then *ordered multi-pushdown automata* [6,1] appear as a special case of multi-pushdown automata.

5.1 Multi-pushdown Automata

Multi-pushdown automata have one read-only left to right input tape and $n \geq 1$ read-write memory tapes (stacks) with a last-in-first-out rewriting policy. A transition is of the form $t = \langle q, \gamma_1, \ldots, \gamma_n \rangle \xrightarrow{a} \langle q', \alpha_1, \ldots, \alpha_n \rangle$. Being in a configuration $(p, w_1, \ldots, w_n)$, which is composed of a state p and a stack content w_i for each stack i, t can be applied if both $q = p$ and the i-th stack is of the form $\gamma_i w'_i$ for some w'_i. Taking the transition and reading the a (which might be the empty word), the system moves to the successor configuration $(q', \alpha_1 w'_1, \ldots, \alpha_n w'_n)$.

Definition 3 (Multi-pushdown automata). *A* multi-pushdown automaton *(MPA) is a tuple $\mathcal{M} = (n, Q, \Sigma, \Gamma, \Delta, q_0, \gamma_0, F)$ where:*

- $n \geq 1$ *is the number of stacks.*
- Q *is the finite non-empty set of* states.
- Σ *is the finite set of* input symbols.
- Γ *is the finite set of* stack symbols *containing the special stack symbol* $\perp$.
- $\Delta \subseteq \big(Q \times (\Gamma_\epsilon)^n\big) \times \Sigma_\epsilon \times \big(Q \times (\Gamma^*)^n\big)$ *is the* transition relation *such that, for all* $((q, \gamma_1, \ldots, \gamma_n), a, (q', \alpha_1, \ldots, \alpha_n)) \in \Delta$ *and* $i \in [1, n]$, *we have:*
  - $|\alpha_i| \leq 2$.
  - *If* $\gamma_i \neq \perp$, *then* $\alpha_i \in (\Gamma \setminus \{\perp\})^*$.
  - *If* $\gamma_i = \perp$, *then* $\alpha_i = \alpha'_i \perp$ *for some* $\alpha'_i \in \Gamma_\epsilon$.
- $q_0 \in Q$ *is the* initial state.
- $\gamma_0 \in (\Gamma \setminus \{\perp\})$ *is the* initial stack symbol.
- $F \subseteq Q$ *is the set of* final states.

The size of $\mathcal{M}$, denoted by $|\mathcal{M}|$, is defined by $(n + |Q| + |\Sigma| + |\Gamma|)$. In the rest of this paper, we use $\langle q, \gamma_1, \ldots, \gamma_n \rangle \xrightarrow{a}_{\mathcal{M}} \langle q', \alpha_1, \ldots, \alpha_n \rangle$ to denote that the transition $((q, \gamma_1, \ldots, \gamma_n), a, (q', \alpha_1, \ldots, \alpha_n))$ is in Δ. Moreover, we denote by $\mathcal{M}(q, \gamma, q')$ the multi-pushdown automaton defined by the tuple $(n, Q, \Sigma, \Gamma, \Delta, q, \gamma, \{q'\})$.

A stack content of $\mathcal{M}$ is a sequence from $\mathit{Stack}(\mathcal{M}) = (\Gamma \setminus \{\perp\})^* \{\perp\}$. A configuration of $\mathcal{M}$ is a $(n+1)$-tuple $(q, w_1, \ldots, w_n)$ with $q \in Q$, and $w_1, \ldots, w_n \in \mathit{Stack}(\mathcal{M})$. A configuration $(q, w_1, \ldots, w_n)$ is final if $q \in F$ and $w_1 = \cdots = w_n = \perp$. The set of configurations of $\mathcal{M}$ is denoted by $\mathit{Conf}(\mathcal{M})$.

The behavior of the MPA $\mathcal{M}$ is described by its corresponding transition system $\mathcal{T}(\mathcal{M})$ defined by the tuple $(\mathit{Conf}(\mathcal{M}), \Sigma, \rightarrow)$ where $\rightarrow$ is the smallest transition relation satisfying the following condition: if $\langle q, \gamma_1, \ldots, \gamma_n \rangle \xrightarrow{a}_{\mathcal{M}} \langle q', \alpha_1, \ldots, \alpha_n \rangle$, then $(q, \gamma_1 w_1, \ldots, \gamma_n w_n) \xrightarrow{a}_{\mathcal{T}(\mathcal{M})} (q', \alpha_1 w_1, \ldots, \alpha_n w_n)$ for all $w_1, \ldots, w_n \in \Gamma^*$ such that $\gamma_1 w_1, \ldots, \gamma_n w_n \in \mathit{Stack}(\mathcal{M})$. Observe that the symbol $\perp$ marks the bottom of a stack. According to the transition relation, $\perp$ can never be popped.

The language accepted (or recognized) by $\mathcal{M}$ is defined by the set $L(\mathcal{M}) = \{\tau \in \Sigma^* \mid (q_0, \gamma_0 \perp, \perp, \ldots, \perp) \xRightarrow{\tau}{}^*_{\mathcal{T}(\mathcal{M})} c$ for some final configuration $c\}$.

5.2 Ordered Multi-pushdown Automata

An ordered multi-pushdown automaton is a multi-pushdown automaton in which one can pop only from the first non-empty stack (i.e., all preceding stacks are equal to $\bot$). In the following, we consider only ordered multi-pushdown automata in normal form with respect to the definitions in [1].

Definition 4 (Ordered multi-pushdown automata). *An ordered multi-pushdown automaton (OMPA for short) is a multi-pushdown automaton* $(n, Q, \Sigma, \Gamma, \Delta, q_0, \gamma_0, F)$ *such that* Δ *contains only the following types of transitions:*

- $\langle q, \gamma, \epsilon, \ldots, \epsilon\rangle \xrightarrow{a}_{\mathcal{M}} \langle q', \gamma''\gamma', \epsilon, \ldots, \epsilon\rangle$ *for some* $q, q' \in Q$*,* $\gamma, \gamma', \gamma'' \in (\Gamma \setminus \{\bot\})$ *and* $a \in \Sigma_\epsilon$*.*
- $\langle q, \gamma, \epsilon, \ldots, \epsilon\rangle \xrightarrow{a}_{\mathcal{M}} \langle q', \epsilon, \ldots, \epsilon, \gamma', \epsilon, \ldots, \epsilon\rangle$ *for some* $q, q' \in Q$*,* $\gamma, \gamma' \in (\Gamma \setminus \{\bot\})$ *and* $a \in \Sigma_\epsilon$ *(*γ' *is pushed on one of stacks* 2 *to* n*).*
- $\langle q, \bot, \ldots, \bot, \gamma, \epsilon, \ldots, \epsilon\rangle \xrightarrow{a}_{\mathcal{M}} \langle q', \gamma'\bot, \bot, \ldots, \bot, \epsilon, \epsilon, \ldots, \epsilon\rangle$ *for some* $q, q' \in Q$*,* $\gamma, \gamma' \in (\Gamma \setminus \{\bot\})$ *and* $a \in \Sigma_\epsilon$ *(*γ *is popped from one of the stacks* 2 *to* n*).*
- $\langle q, \gamma, \epsilon, \ldots, \epsilon\rangle \xrightarrow{a} \langle q', \epsilon, \ldots, \epsilon\rangle$ *for some* $q, q' \in Q$*,* $\gamma \in (\Gamma \setminus \{\bot\})$ *and* $a \in \Sigma_\epsilon$

We introduce the following abbreviations: (1) For $n \geq 1$, we call a MPA/OMPA a n-MPA/n-OMPA, respectively, if its number of stacks is n, and (2) A MPA *over* Σ is a MPA with input alphabet Σ.

Next, we recall some properties of the class of languages recognized by n-OMPA.

Lemma 2 ([6]). *If* $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *are two* n*-OMPAs over an alphabet* Σ*, then it is possible to construct a* n*-OMPA* $\mathcal{M}$ *over* Σ *such that:* (1) $L(\mathcal{M}) = L(\mathcal{M}_1) \cup L(\mathcal{M}_2)$ *and* $|\mathcal{M}| = O(|\mathcal{M}_1| + |\mathcal{M}_2|)$*.*

Lemma 3 ([6]). *Let* Σ *be an alphabet. Given a* n*-OMPA* $\mathcal{M}$ *over* Σ *and a finite state automaton* $\mathcal{A}$ *over* Σ*, then it is possible to construct a* n*-OMPA* $\mathcal{M}'$ *such that:* $L(\mathcal{M}') = L(\mathcal{M}) \cap L(\mathcal{A})$ *and* $|\mathcal{M}'| = O(|\mathcal{M}| \cdot |\mathcal{A}|)$*.*

6 The Emptiness Problem for a n-OMPA Is in 2ETIME

In this section, we show that the emptiness problem for ordered pushdown automata is in 2ETIME. (We provide here a simpler proof of the 2ETIME upper bound than the one given in [1].) To this aim, we proceed as follows:

- First, we show that, given a n-OMPA $\mathcal{M}$ with $n > 1$, it is possible to construct a GPA $\mathcal{P}$ with transition languages defined by $(n-1)$-OMPAs of size $O(|\mathcal{M}|^2)$ such that the emptiness problem of $\mathcal{M}$ can be reduced to the emptiness problem of $\mathcal{P}$. Let us present the main steps of this construction. For that, let us consider an accepting run ρ of $\mathcal{M}$. This run can be seen as a sequence of runs of the form $\varsigma_1\sigma_1\varsigma_2\sigma_2 \cdots \varsigma_m\sigma_m$ such that pop operations are exclusive to the first $(n-1)$-stacks (resp. the n-th stack) of $\mathcal{M}$ during the sequence of runs $\varsigma_1, \varsigma_2, \ldots, \varsigma_m$ (resp. $\sigma_1, \sigma_2, \ldots, \sigma_m$). Observe that, by definition, the first $(n-1)$-stacks of $\mathcal{M}$ are empty along the runs $\sigma_1, \sigma_2, \ldots, \sigma_m$. Moreover, at the beginning of the runs

$\varsigma_1, \varsigma_2, \ldots, \varsigma_m$, the OMPA $\mathcal{M}$ is in some configuration c where the first stack of $\mathcal{M}$ contains just one symbol and the stacks from 2 to $n-1$ are empty (i.e., c of the form $(q, \gamma\bot, \bot, \ldots, \bot, w)$).

Then, we construct $\mathcal{P}$ such that the following invariant is preserved during the simulation of $\mathcal{M}$: The state and the content of the stack of $\mathcal{P}$ are the same as the state and the content of the n-th stack of $\mathcal{M}$ when its first $(n-1)$-stacks are empty (and so, $L(\mathcal{P}) \neq \emptyset$ if and only if $L(\mathcal{M}) \neq \emptyset$). To this aim, a pushdown operation of $\mathcal{M}$ that pops a symbol γ from the n-th stack is simply simulated by a pushdown operation of $\mathcal{P}$ that pops the symbol γ. This implies that a run of the form σ_i, with $1 \leq i \leq m$, that pops the word u_i from the n-th stack of $\mathcal{M}$ is simulated by a run of $\mathcal{P}$ that pops the same word u_i. Now, for every $j \in [1, m]$, we need to compute the pushed word v_j into the n-th stack of $\mathcal{M}$ during the run ς_j in order to be pushed also by $\mathcal{P}$. For that, let $L_{(q,\gamma,q')}$ be the set of all possible pushed words u into the n-th stack of $\mathcal{M}$ by a run $(q, \gamma\bot, \bot, \ldots, \bot, w) \overset{\tau}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{M})} (q', \bot, \bot, \ldots, \bot, uw)$ where pop operations are exclusive to the first $(n-1)$-stacks of $\mathcal{M}$. We show that this language $L_{(q,\gamma,q')}$ can be defined by a $(n-1)$-OMPA $\mathcal{M}'(q, \gamma, q')$ over the stack alphabet of $\mathcal{M}$ that: (1) performs the same operations on its state and $(n-1)$-stacks as the one performed by $\mathcal{M}$ on its state and its first $(n-1)$ stacks while discarding the pop operations of $\mathcal{M}$ over the n^{th} stack, and (2) makes visible as transition labels the pushed symbols over the n^{th} stack of $\mathcal{M}$. Now, to simulate the run $\varsigma_j = (q_j, \gamma_j\bot, \bot, \ldots, \bot, w_j) \overset{\tau_j}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{M})} (q'_j, \bot, \bot, \ldots, \bot, u_j w_j)$ of $\mathcal{M}$, $\mathcal{P}$ can push into its stack the word $u_j \in L(\mathcal{M}'(q_j, \gamma_j, q'_j))$.

- Then, we prove, by induction on n, that the emptiness problem for the n-OMPA $\mathcal{M}$ is in 2ETIME with respect to the number of stacks. For that, we assume that the emptiness problem for $(n-1)$-OMPAs can be solved in 2ETIME. This implies that the GPA $\mathcal{P}$ (that simulates $\mathcal{M}$) is effective (see Definition 2 and Lemma 3). Now, we can use Theorem 2 to prove the decidability of the emptiness problem of the effective GPA $\mathcal{P}$ (and so, of the n-OMPA $\mathcal{M}$). To show that the emptiness problem of $\mathcal{P}$ and $\mathcal{M}$ is in 2ETIME, we estimate the running time of our saturation procedure, given in section 4, under the assumption that the emptiness problem for $(n-1)$-OMPAs can be solved in 2ETIME.

Let us give in more details of the proof described above.

6.1 Simulation of a n-OMPA by a GPA

In the following, we prove that, given an OMPA $\mathcal{M}$, we can construct a GPA $\mathcal{P}$, with transition languages defined by $(n-1)$-OMPAs of size $O(|\mathcal{M}|^2)$, such that the emptiness problem for $\mathcal{M}$ is reducible to the emptiness problem for $\mathcal{P}$.

Theorem 3. *Given an OMPA $\mathcal{M} = (n, Q, \Sigma, \Gamma, \Delta, q_0, \gamma_0, F)$ with $n > 1$, it is possible to construct a GPA $\mathcal{P} = (P, \Sigma', \Gamma, \delta, p_0, \bot, \{p_f\})$ such that $P = Q \cup \{p_0, p_f\}$, $\Sigma' = Q$, and we have:*

- *$L(\mathcal{M}) \neq \emptyset$ if and only if $L(\mathcal{P}) \neq \emptyset$, and*

- *For every* $p_1, p_2 \in P$, $a \in \Sigma'_\epsilon$, *and* $\gamma \in \Gamma$, *there is a* $(n-1)$*-OMPA* $\mathcal{M}_{(p_1,\gamma,a,p_2)}$ *over* Γ *s.t.* $L(\mathcal{M}_{(p_1,\gamma,a,p_2)}) = \big(\delta(p_1,\gamma,a,p_2)\big)^R$ *and* $|\mathcal{M}_{(p_1,\gamma,a,p_2)}| = O(|\mathcal{M}|^2)$.

The proof of Theorem 3 is structured as follows. First, we define a $(n-1)$-OMPA $\mathcal{M}'$ over the alphabet Γ that: (1) performs the same operations on its state and $(n-1)$-stacks as the one performed by $\mathcal{M}$ on its state and its first $(n-1)$ stacks while discarding the pop operations of $\mathcal{M}$ on the n^{th} stack, and (2) makes visible as transition labels the pushed symbols over the n^{th} stack of $\mathcal{M}$. Intuitively, depending on the initial and final configurations of $\mathcal{M}'$, the "language" of $\mathcal{M}'$ summarizes the effect of a sequence of pop operations of $\mathcal{M}$ over the first $(n-1)$-stacks on the n^{th} stack of $\mathcal{M}$. So, if we are interested only by the configurations of $\mathcal{M}$ where the first $(n-1)$ stacks are empty, a run of $\mathcal{M}$ can be seen as a sequence of alternations of a pop operation of $\mathcal{M}$ over the n^{th} stack and a push operation over the n^{th} stack of a word in the "language" of $\mathcal{M}'$.

Then, we construct a generalized pushdown automaton $\mathcal{P}$ such that the state and the stack content of $\mathcal{P}$ are the same as the state and the n^{th}-stack content of $\mathcal{M}$ when the first $(n-1)$ stacks of $\mathcal{M}$ are empty. In the definition of $\mathcal{P}$, we use the $(n-1)$-OMPA $\mathcal{M}'$ to characterize the pushed word on the n^{th} stack of $\mathcal{M}$ due to a sequence of pop operations of $\mathcal{M}$ on the $(n-1)$ first stacks of $\mathcal{M}$. This implies that the emptiness problem for $\mathcal{M}$ is reducible to its corresponding problem for $\mathcal{P}$

Constructing the $(n-1)$**-OMPA** $\mathcal{M}'$**:** Let us introduce the following the n-OMPA $\mathcal{M}_{[1,n[} = (n, Q, \Sigma, \Gamma, \Delta_{[1,n[}, q_0, \gamma_0, F)$ such that $\Delta_{[1,n[} = \Delta \cap \big(Q \times (\Gamma_\epsilon)^{n-1} \times \{\epsilon\}) \times \Sigma_\epsilon \times (Q \times (\Gamma^*)^n)\big)$. Intuitively, $\mathcal{M}_{[1,n[}$ is built up from $\mathcal{M}$ by discarding pop operations of $\mathcal{M}$ over the n^{th} stack. Then, let $\mathcal{M}' = (n-1, Q, \Gamma, \Gamma, \Delta', q_0, \gamma_0, F)$ be the $(n-1)$-OMPA, built out from $\mathcal{M}_{[1,n[}$, which (1) performs the same operations on the first $n-1$ stacks of $\mathcal{M}_{[1,n[}$, and (2) makes visible as transition label the pushed stack symbol over the n^{th} stack of $\mathcal{M}_{[1,n[}$. Formally, Δ' is defined as the smallest transition relation satisfying the following conditions:

- If $\langle q, \gamma_1, \ldots, \gamma_{n-1}, \epsilon\rangle \xrightarrow{a}_{\mathcal{M}_{[1,n[}} \langle q', \alpha_1, \ldots, \alpha_{n-1}, \epsilon\rangle$ for some $q, q' \in Q$, $\gamma_1, \ldots, \gamma_{n-1} \in \Gamma_\epsilon$, $a \in \Sigma_\epsilon$, and $\alpha_1, \ldots, \alpha_{n-1} \in \Gamma^*$, then $\langle q, \gamma_1, \ldots, \gamma_{n-1}\rangle \xrightarrow{\epsilon}_{\mathcal{M}'} \langle q', \alpha_1, \ldots, \alpha_{n-1}\rangle$.
- If $\langle q, \gamma, \epsilon, \ldots, \epsilon\rangle \xrightarrow{a}_{\mathcal{M}_{[1,n[}} \langle q', \epsilon, \ldots, \epsilon, \gamma'\rangle$ for some $q, q' \in Q$, $a \in \Sigma_\epsilon$, and $\gamma, \gamma' \in (\Gamma \setminus \{\bot\})$, then $\langle q, \gamma, \epsilon, \ldots, \epsilon\rangle \xrightarrow{\gamma'}_{\mathcal{M}'} \langle q', \epsilon, \ldots, \epsilon\rangle$.

Let us now give the relation between the effect of a sequence of operations of $\mathcal{M}_{[1,n[}$ on the n^{th}-stack and the language of $\mathcal{M}'$.

Lemma 4. *For every* $q, q' \in Q$, *and* $w_1, w_1', \ldots, w_n, w_n' \in Stack(\mathcal{M}_{[1,n[})$, $(q, w_1, \ldots, w_n) \overset{\tau}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{M}_{[1,n[})} (q', w_1', \ldots, w_n')$ *for some* $\tau \in \Sigma^*$ *if and only if there is* $u \in \Gamma^*$ *such that* $(q, w_1, \ldots, w_{n-1}) \overset{u}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{M}')} (q', w_1', \ldots, w_{n-1}')$ *and* $w_n' = u^R w_n$.

Constructing the GPA $\mathcal{P}$**:** We are ready now to define the generalized pushdown automaton $\mathcal{P} = (P, \Sigma', \Gamma, \delta, p_0, \bot, \{p_f\})$, with $P = Q \cup \{p_0, p_f\}$ and $\Sigma' = Q$, that keeps track of the state and the content of the n-th stack of $\mathcal{M}$ when the first $(n-1)$ stacks are empty. Formally, $\mathcal{P}$ is built from $\mathcal{M}$ as follows: For every $p, p' \in P$, $q \in \Sigma'_\epsilon$, and $\gamma \in \Gamma$, we have:

- If $p = p_0$, $\gamma = \bot$, $q = q_0$, and $p' \in Q$, then $\delta(p, \gamma, q, p') = \{u^R\bot \mid u \in L(\mathcal{M}'(q_0, \gamma_0, p'))\}$.
- If $p \in F$, $\gamma = \bot$, $q = \epsilon$, and $p' = p_f$, then $\delta(p, \gamma, q, p') = \{\epsilon\}$.
- If $p, p' \in Q$, $\gamma \neq \bot$, and $q \in Q$, then $\delta(p, \gamma, q, p') = \bigcup_{\gamma' \in \Gamma'} \big(L(\mathcal{M}'(q, \gamma', p'))\big)^R$ where $\Gamma' = \{\gamma' \in \Gamma \mid \exists a \in \Sigma_\epsilon,\ \langle p, \bot, \ldots, \bot, \gamma\rangle \xrightarrow{a}_{\mathcal{M}} \langle q, \gamma'\bot, \bot, \ldots, \bot, \epsilon\rangle\}$.
- Otherwise, $\delta(p, \gamma, q, p') = \emptyset$.

Observe that for every $p_1, p_2 \in P$, $q \in Q_\epsilon$, and $\gamma \in \Gamma$, we can construct an $(n-1)$-OMPA $\mathcal{M}_{(p_1,\gamma,q,p_2)}$ over Γ such that $L(M_{(p_1,\gamma,q,p_2)}) = \big(\delta(p_1, \gamma, q, p_2)\big)^R$ and $|\mathcal{M}_{(p_1,\gamma,q,p_2)}| = O(|\mathcal{M}|^2)$. This can be easily proved using Lemma 2.

To complete the proof of Theorem 3, it remains to show that the emptiness problem for $\mathcal{M}$ is reducible to its corresponding problem for $\mathcal{P}$. This is stated by Lemma 5.

Lemma 5. *$L(\mathcal{M}) \neq \emptyset$ if and only if $L(\mathcal{P}) \neq \emptyset$.*

6.2 Emptiness of a n-OMPA Is in 2ETIME

In the following, we show that the emptiness problem for a n-OMPA is in 2ETIME.

Theorem 4. *The emptiness problem for a n-OMPA $\mathcal{M}$ can be solved in time $O(|\mathcal{M}|^{2^{dn}})$ for some constant d.*

Proof. Let $\mathcal{M} = (n, Q, \Sigma, \Gamma, \Delta, q_0, \gamma_0, F)$ be a n-OMPA. To prove Theorem 4, we proceed by induction on n.

Basis. $n = 1$. Then, $\mathcal{M}$ is a pushdown automaton. From [4], we know that the emptiness problem for $\mathcal{M}$ can be solved in time polynomial in $|\mathcal{M}|$.

Step. $n > 1$. Then, we can apply Theorem 3 to construct a generalized pushdown automaton $\mathcal{P} = (P, Q, \Gamma, \delta, p_0, \bot, \{p_f\})$, with $P = Q \cup \{p_0, p_f\}$, such that:

- $L(\mathcal{P}) \neq \emptyset$ if and only if $L(\mathcal{M}) \neq \emptyset$, and
- For every $p_1, p_2 \in P$, $a \in Q_\epsilon$, and $\gamma \in \Gamma$, there is a $(n-1)$-OMPA $\mathcal{M}_{(p_1,\gamma,a,p_2)}$ over Γ s.t. $L(M_{(p_1,\gamma,a,p_2)}) = \big(\delta(p_1, \gamma, a, p_2)\big)^R$ and $|\mathcal{M}_{(p_1,\gamma,a,p_2)}| = O(|\mathcal{M}|^2)$.

It is easy to observe that $\mathcal{P}$ is an effective generalized pushdown automaton. This is established by the following lemma.

Lemma 6. *$\mathcal{P}$ is an effective generalized pushdown automaton.*

From Theorem 2, Theorem 3, and Lemma 6, we deduce that the emptiness problem for the n-OMPA $\mathcal{M}$ is decidable. Let us now estimate the running time of the decision procedure. From Theorem 2, we know that the emptiness problem of $\mathcal{P}$ is reducible to compute the set of predecessors of the configuration (p_f, ϵ) since $L(\mathcal{P}) \neq \emptyset$ if and only if $(p_0, \bot) \in Pre^*_{\mathcal{T}(\mathcal{P})}(\{p_f\} \times \{\epsilon\})$.

Let $\mathcal{A}$ be the $\mathcal{P}$-automaton that recognizes the configuration (p_f, ϵ) of $\mathcal{P}$. It is easy to see that such $\mathcal{P}$-automaton $\mathcal{A}$, with $|\mathcal{A}| = O(|\mathcal{M}|)$, is effectively constructible. Now, we need to analysis the running time of the saturation procedure (given in section 4) applied to $\mathcal{A}$. For that, let $\mathcal{A}_0, \ldots, \mathcal{A}_i$ be the sequence of $\mathcal{P}$-automaton obtained from the saturation procedure such that $\mathcal{A}_0 = \mathcal{A}$ and $L_{\mathcal{P}}(\mathcal{A}_i) = Pre^*_{\mathcal{T}(\mathcal{P})}(L_{\mathcal{P}}(\mathcal{A}))$. Then, we have $i = O(|\mathcal{M}|^3)$ since the number of possible new transitions of $\mathcal{A}$ is finite. Moreover, at each step j, with $0 \leq j \leq i$, we need to check, for every $q \in Q_{\mathcal{A}}$, $p, p' \in P$, $\gamma \in \Gamma$, and $a \in Q_\epsilon$, whether $L(\mathcal{A}_j)(\{p'\}, \{q\}) \cap \delta(p, \gamma, a, p') \neq \emptyset$.

Using Lemma 3, we can construct, in time polynomial in $|\mathcal{M}|$, a $(n-1)$-OMPA $\mathcal{M}'_{(q,p,\gamma,a,p')}$ such that $L(\mathcal{M}'_{(q,p,\gamma,a,p')}) = (L(\mathcal{A}_j)(\{p'\}, \{q\}))^R \cap L(M_{(p,\gamma,a,p')})$ and $|\mathcal{M}'_{(q,p,\gamma,a,p')}| \leq c(|\mathcal{M}|^3)$ for some constant c. Now, we can apply the induction hypothesis to $\mathcal{M}'$, and we obtain that the problem of checking whether $L(\mathcal{M}'_{(q,p,\gamma,a,p')}) \neq \emptyset$ can be solved in time $O\big((c\,|\mathcal{M}|^3)^{2^{d(n-1)}}\big)$. Putting together all these equations, we obtain that the problem of checking whether $(p_0, \bot) \in Pre^*_{\mathcal{T}(\mathcal{P})}(\{p_f\} \times \{\epsilon\})$ can be solved in time $O\big(|\mathcal{M}|^3|\mathcal{M}|^5(c\,|\mathcal{M}|^3)^{2^{d(n-1)}}\big)$. By taking a constant d as big as needed, we can show that the problem of checking whether $L(\mathcal{M}) \neq \emptyset$ can be solved in time $O(|\mathcal{M}|^{2^{dn}})$. □

7 Bounded-Phase Visibly Multi-Pushdown Automata

In this section, we recall the definition of visibly multi-pushdown automata [16], another subclass of MPA, where an action is associated with a particular stack operation. An action can be a push, pop, or internal action. Formally, a visibly multi-pushdown automaton (VMPA for short) is a tuple $\mathcal{V} = (\mathcal{M}, type)$ where $\mathcal{M} = (n, Q, \Sigma, \Gamma, \Delta, q_0, \gamma_0, F)$ is a MPA and $type : \Sigma \rightarrow (\{Push, Pop\} \times [1,n]) \cup \{Int\}$ is a function satisfying, for all transitions $\langle q, \gamma_1, \ldots, \gamma_n \rangle \xrightarrow{a}_{\mathcal{M}} \langle q', \alpha_1, \ldots, \alpha_n \rangle$:

- $a \neq \epsilon$,
- if $type(a) = (Push, i)$ for some $i \in [1,n]$, then $\gamma_1 = \ldots = \gamma_n = \epsilon$, $\alpha_i \in (\Gamma \setminus \{\bot\})$, and $\alpha_j = \epsilon$ for all $j \in [1,n] \setminus \{i\}$,
- if $type(a) = (Pop, i)$ for some $i \in [1,n]$, then $\gamma_i \in \Gamma$, $\alpha_i \in \{\bot\} \cup \{\epsilon\}$, and $\gamma_j = \alpha_j = \epsilon$ for all $j \in [1,n] \setminus \{i\}$, and
- if $type(a) = Int$, then $\gamma_i = \alpha_i = \epsilon$ for all $i \in [1,n]$.

If, in a VMPA, we restrict the number of phases, where in one phase pop operations are exclusive to one stack, then we obtain bounded-phase visibly multi-pushdown automata. A bounded-phase visibly multi-pushdown automaton (BVMPA for short) is a triple $\mathcal{B} = (\mathcal{M}, type, \tau)$ where $\mathcal{M} = (n, Q, \Sigma, \Gamma, \Delta, q_0, \gamma_0, F)$ is a MPA, τ is a word over $[1,n]$, and $(\mathcal{M}, type)$ is a VMPA. For every $i \in [1,n]$, let $\Sigma_i = \{a \in \Sigma \,|\, \nexists j \neq i, type(a) = (Pop, j)\}$. The language accepted by $\mathcal{B}$ is defined as follows: $L(\mathcal{B}) = L(\mathcal{M}) \cap (\Sigma^*_{\tau(1)} \Sigma^*_{\tau(2)} \cdots \Sigma^*_{\tau(|\tau|)})$

In the rest of this paper, we use a τ-phase n-BVMPA over Σ to denote a BVMPA of the form $(\mathcal{M}, type, \tau)$ with $\mathcal{M}$ is a n-MPA over Σ.

Next, we show that the class of languages accepted bounded-phase visibly multi-pushdown automata is closed under intersection with a regular languages.

Lemma 7. *Let Σ be an alphabet. Given a τ-phase n-BVMPA $\mathcal{B} = (\mathcal{M}, type, \tau)$ over Σ and a FSA $\mathcal{A}$ over Σ, it is possible to construct a τ-phase n-BVMPA $\mathcal{B}' = (\mathcal{M}', type, \tau)$ such that $L(\mathcal{B}') = L(\mathcal{B}) \cap L(\mathcal{A})$ and $|\mathcal{M}'| = O(|\mathcal{M}||\mathcal{A}|)$.*

Proof. Similar to the proof of Lemma 3. □

8 The Emptiness Problem for a τ-Phase n-BVMPA Is in 2ETIME

In this section, we show that the emptiness problem for bounded-phase visibly multi-pushdown automata is in 2ETIME with respect to the length of τ. To this aim, we proceed as follows: First, we prove that the emptiness for a τ-phase n-BVMPA is reducible to the emptiness problem for a generalized pushdown automaton with transition languages defined by τ'-phase n-BVMPAs with $\tau = \tau' \cdot \tau(|\tau|)$. Then, we use the saturation procedure, given in Section 4 to prove, by induction on $|\tau|$, that the emptiness problem for a τ-phase n-BVMPA is in 2ETIME.

For the rest of this section, let us fix a BVMPA $\mathcal{B} = (\mathcal{M}, type, \tau)$ where $\mathcal{M} = (n, Q, \Sigma, \Gamma, \Delta, q_0, \gamma_0, F)$ is a MPA. We assume w.l.o.g that $\Sigma \cap \Gamma = \emptyset$. For every $i \in [1, n]$, let $\Sigma_i = \{a \in \Sigma \mid \nexists j \neq i, type(a) = (Pop, j)\}$. Moreover, let us assume that $k = \tau(|\tau|)$ and $\tau = \tau' k$.

8.1 Simulation of a τ-Phase n-BVMPA by a GPA

In the following, we prove that it is possible to construct a GPA $\mathcal{P}$ such that the emptiness problem for $\mathcal{B}$ is reducible to the emptiness problem for $\mathcal{P}$.

Theorem 5. *Assume that $k > 1$. Then, it is possible to construct a GPA $\mathcal{P} = (P, \Sigma, \Gamma, \delta, p_0, \bot, \{p_f\})$ such that $P = Q \cup \{p_0, p_f\}$ and we have:*

- *$L(\mathcal{B}) \neq \emptyset$ if and only if $L(\mathcal{P}) \neq \emptyset$, and*
- *For every $p, p' \in P$, $a \in \Sigma_\epsilon$, and $\gamma \in \Gamma$, there is a τ'-phase n-BVMPA $\mathcal{B}_{(p,\gamma,a,p')} = (\mathcal{M}_{(p,\gamma,a,p')}, type', \tau')$ over Σ' such that $\Gamma \subseteq \Sigma'$, $|\mathcal{M}_{(p,\gamma,a,p')}| = O(|\mathcal{M}|^2)$, and $\delta(p, \gamma, a, p') = \big((L(\mathcal{B}_{(p,\gamma,a,p')}))|_\Gamma)\big)^R\{\bot\}$.*

Proof. By definition, $L(\mathcal{B}) \neq \emptyset$ if and only if there are $q \in F$ and $\sigma_j \in \Sigma^*_{\tau(j)}$ for all $j \in [1, |\tau|]$ such that $\rho = (q_0, \gamma_0\bot, \bot, \ldots, \bot) \xRightarrow{\sigma_1 \cdots \sigma_{|\tau|}}{}^*_{\mathcal{T}(\mathcal{M})} (q, \bot, \ldots, \bot)$. Thus, the emptiness problem for $\mathcal{B}$ can be reduced to the problem of checking whether there are $q' \in Q$, $w_1, \ldots, w_n \in Stack(\mathcal{M})$, $q \in F$, and $\sigma_j \in \Sigma^*_{\tau(j)}$ for all $j \in [1, |\tau|]$ such that: (1) $w_l = \bot$ for all $l \in [1, n]$ and $l \neq k$, (2) $\rho_1 = (q_0, \gamma_0\bot, \bot, \ldots, \bot) \xRightarrow{\sigma_1 \cdots \sigma_{(|\tau|-1)}}{}^*_{\mathcal{T}(\mathcal{M})} (q', w_1, \ldots, w_n)$, and (3) $\rho_2 = (q', w_1, \ldots, w_n) \xRightarrow{\sigma_{(|\tau|)}}{}^*_{\mathcal{T}(\mathcal{M})} (q, \bot, \ldots, \bot)$. Observe that at the configuration $(q', w_1, \ldots, w_n)$ only the content of the k-stack of $\mathcal{M}$ can be different from $\bot$. Moreover, during the run ρ_2, pop and push operations are exclusive to the k-th stack of $\mathcal{M}$. So, in order to prove Theorem 5, it is sufficient to show that all possible contents w_k of the k-stack of $\mathcal{M}$ reached by the run ρ_1 can be characterized by a language accepted by a τ'-phase n-BVMPA $\mathcal{B}_{q'} = (\mathcal{M}_{q'}, \mathit{type}', \tau')$ over Σ' such that $\Gamma \subseteq \Sigma'$ (i.e., $w_k \in \big((L(\mathcal{B}_{q'})|_{\Gamma})\big)^R\{\bot\}$). Once this is done, we can construct a GPA $\mathcal{P}$ that simulates $\mathcal{B}$. The automaton $\mathcal{P}$ proceeds as follows: First, it pushes in its stack the content w_k of the k-th stack of $\mathcal{M}$ reached by the run ρ_1 using the language of $\mathcal{B}_{q'}$. Then, $\mathcal{P}$ starts to simulate the run ρ_2 by performing the same operations on its state and its stack as the ones performed by $\mathcal{M}$ on its state and its k-th stack. This is possible since along the run ρ_2 pop and push operations of $\mathcal{M}$ are exclusive to k-th stack. Finally, $\mathcal{P}$ checks if the current configurations is final and moves its state to the final state p_f.

Constructing the τ'-phase n-BVMPA $\mathcal{B}_{q'}$: In the following, we show that it is possible to construct, for every $q' \in Q$, a τ'-phase n-BVMPA $\mathcal{B}_{q'}$ such that $w_k \in \big((L(\mathcal{B}_{q'})|_{\Gamma})\big)^R\{\bot\}$ if and only if there are $\sigma_j \in \Sigma^*_{\tau(j)}$ for all $j \in [1, |\tau|[$ such that $(q_0, \gamma_0\bot, \bot, \ldots, \bot) \xRightarrow{\sigma_1 \cdots \sigma_{(|\tau|-1)}}{}^*_{\mathcal{T}(\mathcal{M})} (q', w_1, \ldots, w_n)$ where $w_l = \bot$ for all $l \in [1, n]$ and $l \neq k$. For that, let us define the n-MPA $\mathcal{M}'$ that contains all the transitions of $\mathcal{M}$ and that have the ability to push the new fresh symbol $\sharp$ instead of any possible pushed symbol by $\mathcal{M}$ into the k-th stack. Moreover, the symbol $\sharp$ can be popped from the k-th stack at any time by $\mathcal{M}'$ without changing its state. Formally, $\mathcal{M}'$ is defined by the tuple $(n, Q, \Sigma', \Gamma', \Delta', q_0, \gamma_0, F)$ where $\Gamma' = \Gamma \cup \{\sharp\}$ is the stack alphabet, $\Sigma' = \Sigma \cup \Gamma \cup \{\sharp\}$ is the input alphabet, and Δ' is the smallest transition relation satisfying the following conditions:

- $\Delta \subseteq \Delta'$.
- If $\langle q_1, \gamma_1, \ldots, \gamma_n\rangle \xrightarrow{a}_{\mathcal{M}} \langle q_2, \alpha_1, \ldots, \alpha_n\rangle$ and $\mathit{type}(a) = (\mathit{Push}, k)$, then $\langle q_1, \gamma_1, \ldots, \gamma_n\rangle \xrightarrow{\alpha_k}_{\mathcal{M}_{q'}} \langle q_2, \alpha'_1, \ldots, \alpha'_n\rangle$ with $\alpha'_l = \alpha_l$ for all $l \neq k$ and $\alpha'_k = \sharp$.
- For every $q' \in Q$, $\langle q', \gamma_1, \ldots, \gamma_n\rangle \xrightarrow{\sharp}_{\mathcal{M}_{q'}} \langle q', \alpha'_1, \ldots, \alpha'_n\rangle$ if $\gamma_l = \epsilon$ for all $l \neq k$, $\gamma_k = \sharp$, and $\alpha_1 = \ldots = \alpha_n = \epsilon$.

Let $\mathcal{B}' = (\mathcal{M}', \mathit{type}', \tau')$ be a τ'-phase n-BVMPA where the function $\mathit{type}' : \Sigma' \to (\{\mathit{Push}, \mathit{Pop}\} \times [1, n]) \cup \{\mathit{Int}\}$ is defined as follows: $\mathit{type}'(a) = \mathit{type}(a)$ for all $a \in \Sigma$, $\mathit{type}'(\sharp) = (\mathit{Pop}, k)$, and $\mathit{type}'(\gamma) = (\mathit{Push}, k)$ for all $\gamma \in \Gamma$.

Let $\Xi_k = \{a \in \Sigma' \mid \mathit{type}'(a) = (\mathit{Pop}, k)\}$. Then, from Lemma 7, it is possible to construct, for every q', a τ'-phase n-BVMPA $\mathcal{B}_{q'} = (\mathcal{M}'_{q'}, \mathit{type}', \tau')$ from $\mathcal{B}'$ such that $L(\mathcal{B}_{q'}) = L((\mathcal{M}'(q_0, \gamma_0, q'), \mathit{type}', \tau')) \cap (\Sigma \cup \Gamma)^*\{\sharp\}^*(\Sigma' \setminus \Xi_k)^*$. The relation between $\mathcal{B}_{q'}$ and $\mathcal{M}$ is given by Lemma 8 which can be proved by induction.

Lemma 8. *For every $q' \in Q$, $w_k \in ((L(\mathcal{B}_{q'})|_{\Gamma})^R\{\bot\})$ iff there are $\sigma_j \in \Sigma^*_{\tau(j)}$ for all $j \in [1, |\tau|[$ such that $(q_0, \gamma_0\bot, \bot, \ldots, \bot) \xRightarrow{\sigma_1 \cdots \sigma_{(|\tau|-1)}}{}^*_{\mathcal{T}(\mathcal{M})} (q', w_1, \ldots, w_n)$ where $w_l = \bot$ for all $l \in [1, n]$ and $l \neq k$.*

Constructing the GPA $\mathcal{P}$: Formally, the transition function δ is defined as follows: for every $p, p' \in P$, $\gamma \in \Gamma$, and $a \in \Sigma$, we have:

- Initialization: $\delta(p, \gamma, a, p') = \big(L(\mathcal{B}_{p'})|_{\Gamma}\big)\{\bot\}$ if $p = q_0$, $p' \in Q$, $\gamma = \bot$, and $a = \epsilon$. This transition pushes in the stack of $\mathcal{P}$ the content of the k-th stack of $\mathcal{M}$ reached by the run ρ_1.

- Simulation of a push operation on the k-th stack of $\mathcal{M}$: $\delta(p,\gamma,a,p') = \{\alpha_k\gamma \mid \langle p,\gamma_1,\ldots,\gamma_n\rangle \xrightarrow{a}_{\mathcal{M}} \langle p',\alpha_1,\ldots,\alpha_n\rangle\}$ if $p,p' \in Q$, and $a \in \Sigma_k$ such that $type(a) = (Push, k)$.
- Simulation of a pop operation on the k-th stack of $\mathcal{M}$: $\delta(p,\gamma,a,p') = \{\alpha_k \mid \langle p,\gamma_1,\ldots,\gamma_n\rangle \xrightarrow{a}_{\mathcal{M}} \langle p',\alpha_1,\ldots,\alpha_n\rangle, \gamma = \gamma_k\}$ if $p,p' \in Q$ and $a \in \Sigma_k$ such that $type(a) = (Pop, k)$.
- Simulation of an internal operation of $\mathcal{M}$: $\delta(p,\gamma,a,p') = \{\gamma\}$ if $p,p' \in Q$ and $a \in \Sigma_k$ such that $type(a) = Int$.
- Final configuration: $\delta(p,\gamma,a,p') = \{\epsilon\}$ if $p \in F$, $p' = p_f$, $\gamma = \bot$, and $a = \epsilon$.
- Otherwise: $\delta(p,\gamma,a,p') = \emptyset$.

Then, it is easy to see that $L(\mathcal{B}) \neq \emptyset$ if and only if $L(\mathcal{P}) \neq \emptyset$. □

As an immediate consequence of Theorem 5, we obtain that the emptiness problem for a τ-phase n-BVMPA is in 2ETIME.

Theorem 6. *The emptiness problem for a BVMPA $\mathcal{B} = (\mathcal{M}, type, \tau)$ can be solved in time $O(|\mathcal{M}|^{2^{d|\tau|}})$ for some constant d.*

The proof of Theorem 6 is similar to the proof of Theorem 4.

9 Conclusion

In this paper, we have shown that the emptiness problem for both OMPA and BVMPA can be reduced to the one for the class of effective generalized pushdown automata. We provide here simple algorithms for checking the emptiness problem for each of these models and proving their 2ETIME upper bounds.

Recently, A. Seth has showed in [15] that the set of predecessors of a regular set of configurations of a BVMPA is a regular set and effectively constructible. We believe that our automata-based saturation procedure for computing the set of predecessors for an effective GPA can be used to show (by induction on the number of stacks) that the set of predecessors of a regular set of configurations of an OMPA is a regular set and effectively constructible (which may answer to a question raised in [15]).

It is quite easy to see that the model-checking problems for omega-regular properties of effective generalized pushdown automata are decidable. This can be done by adapting the construction given in [4]. This result can be used to establish some decidability/complexity results concerning the model-checking problems for omega-regular properties of both OMPA and BVMPA.

References

1. Atig, M.F., Bollig, B., Habermehl, P.: Emptiness of multi-pushdown automata is 2ETIME-complete. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 121–133. Springer, Heidelberg (2008)
2. Atig, M.F., Bouajjani, A., Touili, T.: On the reachability analysis of acyclic networks of pushdown systems. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 356–371. Springer, Heidelberg (2008)

3. Bouajjani, A., Esparza, J.: Rewriting models of boolean programs. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 136–150. Springer, Heidelberg (2006)
4. 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)
5. Bouajjani, A., Müller-Olm, M., Touili, T.: Regular symbolic analysis of dynamic networks of pushdown systems. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 473–487. Springer, Heidelberg (2005)
6. Breveglieri, L., Cherubini, A., Citrini, C., Crespi Reghizzi, S.: Multi-push-down languages and grammars. International Journal of Foundations of Computer Science 7(3), 253–292 (1996)
7. Caucal, D.: On infinite transition graphs having a decidable monadic theory. In: Meyer auf der Heide, F., Monien, B. (eds.) ICALP 1996. LNCS, vol. 1099, pp. 194–205. Springer, Heidelberg (1996)
8. Esparza, J., Knoop, J.: An automata-theoretic approach to interprocedural data-flow analysis. In: Thomas, W. (ed.) FOSSACS 1999. LNCS, vol. 1578, pp. 14–30. Springer, Heidelberg (1999)
9. Kahlon, V.: Boundedness vs. unboundedness of lock chains: Characterizing decidability of pairwise cfl-reachability for threads communicating via locks. In: LICS, pp. 27–36. IEEE Computer Society, Los Alamitos (2009)
10. Lal, A., Reps, T.W.: Reducing concurrent analysis under a context bound to sequential analysis. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 37–51. Springer, Heidelberg (2008)
11. Qadeer, S.: The case for context-bounded verification of concurrent programs. In: Havelund, K., Majumdar, R., Palsberg, J. (eds.) SPIN 2008. LNCS, vol. 5156, pp. 3–6. Springer, Heidelberg (2008)
12. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
13. Ramalingam, G.: Context-sensitive synchronization-sensitive analysis is undecidable. ACM Trans. Program. Lang. Syst. 22(2), 416–430 (2000)
14. Reps, T.W., Schwoon, S., Jha, S.: Weighted pushdown systems and their application to interprocedural dataflow analysis. In: Cousot, R. (ed.) SAS 2003. LNCS, vol. 2694, pp. 189–213. Springer, Heidelberg (2003)
15. Seth, A.: Global reachability in bounded phase multi-stack pushdown systems. In: CAV 2010. LNCS. Springer, Heidelberg (2010)
16. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: Proceedings of LICS, pp. 161–170. IEEE, Los Alamitos (2007)
17. La Torre, S., Madhusudan, P., Parlato, G.: An infinite automation characterization of double exponential time. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 33–48. Springer, Heidelberg (2008)
18. La Torre, S., Madhusudan, P., Parlato, G.: Reducing context-bounded concurrent reachability to sequential reachability. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 477–492. Springer, Heidelberg (2009)

A Geometric Approach to the Problem of Unique Decomposition of Processes*

Thibaut Balabonski[1] and Emmanuel Haucourt[2]

[1] Laboratoire PPS, Université Paris Diderot and CNRS, UMR 7126
thibaut.balabonski@pps.jussieu.fr
[2] CEA, LIST, Gif-sur-Yvette, F-91191, France
emmanuel.haucourt@cea.fr

Abstract. This paper proposes a geometric solution to the problem of prime decomposability of concurrent processes first explored by R. Milner and F. Moller in [MM93]. Concurrent programs are given a geometric semantics using cubical areas, for which a unique factorization theorem is proved. An effective factorization method which is correct and complete with respect to the geometric semantics is derived from the factorization theorem. This algorithm is implemented in the static analyzer ALCOOL.

Keywords: Concurrency, Milner problem, Decomposition of processes, Geometric semantics.

1 Introduction: Parallel Programming Problem

This paper aims at introducing some new static analysis technology for concurrent programs. The work presented here gives a new insight into the problem of decomposition of processes, which was first explored by *R. Milner* and *F. Moller* in [MM93]. The main new results are an algorithm maximally decomposing concurrent programs into independent processes (Section 4) and the proof that this prime decomposition is unique in the considered class of programs (Theorem 2). They are derived from a study of algebraic properties of cubical areas.

Given an associative and commutative operator $\|$ for parallel composition of two processes (with the empty process as unit), decomposing a concurrent program P into a multiset $\{P_1, ..., P_n\}$ such that $P = P_1\|...\|P_n$ and the P_is are independent has several interests. For instance the decomposition may be relevant for the allocation of processors to subprograms. Another important concern is the static analysis of concurrent programs, whose complexity grows exponentially with the number of concurrent processes: finding independent subprograms that can be analyzed separately could dramatically decrease the global complexity

* This work has been partially supported by *Agence Nationale pour la Recherche* via the project PANDA (Parallel and Distributed Analysis) ANR-09-BLAN-0169-02.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 132–146, 2010.

of the static analysis. Hence this paper aims at finding the finest decomposition (and proving its uniqueness) for a wide class of concurrent programs.

Let us first take a look at a nontrivial example of independent processes, in the so-called *PV* language introduced by *E.W. Dijkstra* [Dij68] as a simple framework for the study of concurrency with shared resources. The only instructions are $P(name)$ and $V(name)$[1], where *name* is an identifier which refers to a resource. The idea is to have some common pool of resources which can be taken (with P) and released (with V) by concurrent processes. The resources are formalized by semaphores which, depending on their arity, can be held simultaneously by a certain number of processes (arity n allows at most $n-1$ simultaneous processes).

Now suppose c is the name of a ternary semaphore, which means it can be held by at most two processes, and a, b are the names of binary semaphores, also called *mutex* for *mutual exclusion.*

Example 1.

$$\begin{aligned} \Sigma := \pi_1 &= Pa.Pc.Vc.Va \\ \| \ \ \pi_2 &= Pb.Pc.Vc.Vb \\ \| \ \ \pi_3 &= Pa.Pc.Vc.Va \\ \| \ \ \pi_4 &= Pb.Pc.Vc.Vb \end{aligned}$$

A naive syntactic analysis would stamp this program as indecomposable since all processes share the resource c, but the following finer analysis can be made: thanks to mutex a (respectively b), the processes π_1 and π_3 (respectively π_2 and π_4) cannot both hold an occurrence of the resource c at the same time. Then there are never more than two simultaneous requests over c, which means that the instructions Pc and Vc play actually no role in determining the semantics of the program. And without c, Σ can be split in two independent systems (they use disjoint resources). Basically, this example is based on the fact that semaphores are not the real resources, but mere devices used to guard their access. And it may be that some guards are redundant.

This work is based on a geometric semantics for concurrency. The semantics for *PV* programs was implicitly given in [Dij68], then explicited by *Carson* et al.[CR87]. Roughly speaking, the instructions of a process are pinned upon a 1-dimensional "directed" shape, in other words track along which the instructions of the program to execute are written. If N sequential processes run together, one can consider their N instruction pointers as a multidimensional control point.

Although we have made the construction explicit for *PV* programs only, the result applies to any synchronisation or communication mechanism whose geometric interpretation is a so-called *cubical area* (the notion is formalized in Section 3.5). See for instance [GH05] for the geometric semantics of synchronisation barriers, monitors and synchronous or asynchronous communications (with finite or infinite message queues): their geometrical shape is the complement of an orthogonal polyhedron [BMP99, Tha09], which is a special case of cubical area.

[1] P and V stand for the dutch words "Pakken" (take) and "Vrijlaten" (release).

Outline of the paper

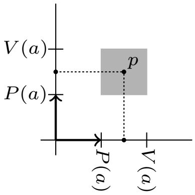
The paper is organized as follows. Section 2 provides the mathematics of the geometric semantics, detailed for PV programs. Section 3 establishes the link between algebraic properties of the semantics and independence of subprograms, and then states and proves prime decomposability theorems for algrebraic frameworks encompassing the geometric semantics (Theorems 1 and 2). Section 4 describes the corresponding algorithm and implementation as well as a detailed example and some benchmarks.

2 The Geometric Semantics

The geometric semantics of a PV program is a subset of the finite dimensional real vector space whose dimension is the number N of processes running concurrently: then each process is associated with a coordinate of $\mathbb{R}^N$. Yet given a mutex `a`, the instructions `P(a)` and `V(a)` that occur in the k^{th} process should be understood as opening and closing parentheses or more geometrically as the least upper bound and the greatest lower bound of an interval I_k of $\mathbb{R}$. The forbidden area generated by a mutex `a` is thus the finite union of hyperrectangles[2] of the following form (with $k < k'$)

$$\underbrace{\mathbb{R}^+ \times \cdots \times \mathbb{R}^+ \times I_k \times \mathbb{R}^+ \times \cdots \times \mathbb{R}^+ \times I_{k'} \times \mathbb{R}^+ \times \cdots \times \mathbb{R}^+}_{\text{product of } N \text{ terms}}$$

For example, `P(a).V(a)` $\|$ `P(a).V(a)` is a program written in PV language. Assuming that a is a mutex (semaphore of arity 2), its geometric model is $(\mathbb{R}^+)^2 \backslash [1,2[^2$. Intuitively, a point p in $[1,2[^2$ would correspond to the situation where both processes hold the semaphore a, which is forbidden by the semantics of mutices.



In the sequel of this section we formalize the PV language syntax as well as the construction of the geometric semantics.

Denote the positive half-line $[0, +\infty[$ by $\mathbb{R}^+$. For each $\alpha \in \mathbb{N}\backslash\{0,1\}$ let S_α be an infinite countable set whose elements are the semaphores of arity α of the PV language. A **PV process** is a finite sequence on the alphabet

$$A := \{P(s), V(s) \mid s \in \bigcup_{\alpha \geq 2} S_\alpha\}$$

and a **PV program** is a finite (and possibly empty) multiset of PV processes. The parallel operator then corresponds to the multiset addition therefore it is associative and commutative[3]. Given a semaphore s and a process π, the se-

[2] However we will more likely write "cube" instead.

[3] The collection of multisets over a set $\mathbb{A}$ forms a monoid which is isomorphic to the free commutative monoid over $\mathbb{A}$. The first terminology is usually used by computer scientists while mathematicians prefer the second one. Anyway it will be described and caracterized in the Section 3.

quences $(x_k)_{k\in\mathbb{N}}$ and $(y_k)_{k\in\mathbb{N}}$ are recursively defined as follows: set $y_{-1} = 0$ and

- $x_k = \min\{n \in \mathbb{N} \mid n > y_{k-1} \text{ and } \pi(n) \text{ is } P(s)\}$
- $y_k = \min\{n \in \mathbb{N} \mid n > x_k \text{ and } \pi(n) \text{ is } V(s)\}$

with the convention that $\min \emptyset = \infty$, $\pi(n)$ denotes the n^{th} term of the process π and its first term is $\pi(1)$. Then, the **busy area** of s in π is[4]

$$B_s(\pi) := \bigcup_{k\in\mathbb{N}} [x_k, y_k[$$

Actually this description implies some extra assumptions upon the way instructions are interpreted. Namely a process cannot hold more than one occurrence of a given resource. Thus a process already holding an occurrence of a semaphore s ignores any instruction $P(s)$, and similarly a process holding no occurrence of s ignores any instruction $V(s)$. Then denote by $\chi_s^\pi : \mathbb{R} \to \mathbb{R}$ the characteristic function of B_s defined by

$$\chi_s^\pi(x) = \begin{cases} 1 \text{ if } x \in B_s(\pi) \\ 0 \text{ otherwise} \end{cases}$$

Because the sequence π is finite, there exists some k such that $x_k = \infty$ and for any such k and any $k' \geq k$, one also has $x_{k'} = \infty$. In particular, if the instruction $P(s)$ does not appear in π, then $B_s(\pi)$ is empty and χ_s^π is the null map. The geometric model of a PV program with N processes running concurrently is a subpospace of $[0, +\infty[^N$ defined as follows:
- Call $\Pi = (\pi_1, \ldots, \pi_N)$ the program to model.
- Given a semaphore s of arity α define the **forbidden area** of s in Π as

$$F_s := \left\{\overrightarrow{x} \in [0, +\infty[^N \;\middle|\; \overrightarrow{\chi_s} \cdot \overrightarrow{x} \geq \alpha\right\}$$

where $\overrightarrow{x} = (x_1, \ldots, x_N)$, $\overrightarrow{\chi_s} = (\chi_s^{\pi_1}, \ldots, \chi_s^{\pi_N})$ and $\overrightarrow{\chi_s} \cdot \overrightarrow{x} = \sum_{i=1}^{N} \chi_s^{\pi_i}(x_i)$. The value $\overrightarrow{\chi_s} \cdot \overrightarrow{x}$ indicates how many occurrences of the semaphore s are held when the instruction pointer is at position $\overrightarrow{x}$. Note that F_s is a finite union of hyperrectangles which may be empty even if s appears in the program Π. In the end, the **forbidden area** of the program Π is the following union over S the union of all the sets S_α.

$$F := \bigcup_{s\in S} F_s$$

Because there are finitely many resource names s appearing in a PV program, there are finitely many nonempty sets F_s. Hence the previous union is still a finite union of hyperrectangles. The **state space** or **geometric model** of Π is

[4] Including the greatest lower bound and removing the least upper bound is the mathematical interpretation of the following convention: the changes induced by an instruction are effective exactly when the instruction pointer reaches it.

then $[0,+\infty[^N \backslash F$, and is denoted by $[\![\Pi]\!]$. Remark that the geometric model is also a finite union of hyperrectangles.

In other words, the state space of Π is the set of positions of the "multi-dimensional instruction pointer" for which the number of occurrences of each semaphore s is strictly below its arity α. If Π is made of N concurrent process, this space is a N-dimensional euclidean space with (cubical) holes. As an example, Figure 1 shows the construction of the geometric model of the PV program $P(a)P(b)V(b)V(a) \parallel P(b)P(a)V(a)V(b)$ (refered to as the *swiss flag*). Figure 2 gives a simplified version of Example 1 fitting in three dimensions.

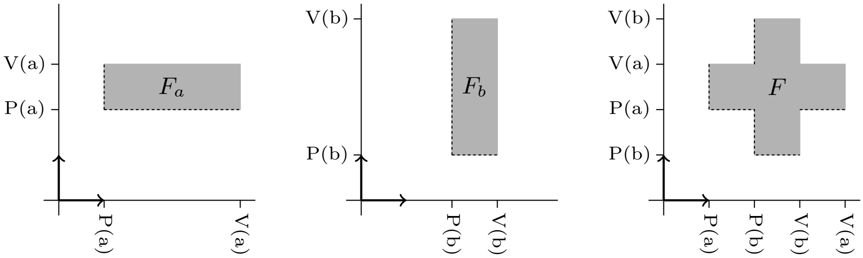
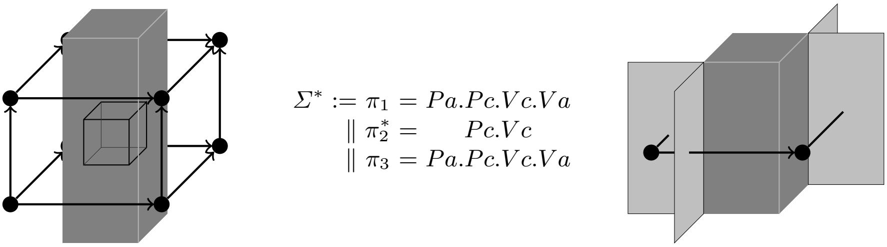


Fig. 1. Construction of a geometric model: the swiss flag



$$\begin{aligned} \Sigma^* := \pi_1 &= Pa.Pc.Vc.Va \\ \parallel \pi_2^* &= \quad\; Pc.Vc \\ \parallel \pi_3 &= Pa.Pc.Vc.Va \end{aligned}$$

Fig. 2. Example in three dimensions

Intuitively, the graphs pictured here correspond to the *essential components* of the state space, see [GH07] for developments on this topic. The little cube on the left picture is the forbidden area of the semaphore c, which is contained in the forbidden area of the mutex a (in the full –and 4D– example Σ the forbidden area of c is contained in the union of the forbidden areas of a and b).

3 The Problem of Unique Decomposition

Now that the geometric semantics of programs is defined, let us refocus on the main goal: finding the independent parts of a concurrent program. Hence the question: what does independence mean in this geometrical setting?

3.1 Parallel Composition vs. Cartesian Product

A general definition has to be given for **independence**: say a program Π is independent from another program Π' when its behaviour is unaffected by parallel composition with Π', whatever the way Π' is executed. That means, the presence of Π', as well as its instruction pointer, has no effect on the semantics of Π. A geometric translation of this assertion is: in the geometric model of $\Pi \| \Pi'$, the cylinder[5] over any state of Π' (i.e. the subspace of all points with given fix coordinates for the Π' component) is equal to the model of Π.

Hence two programs Π and Π' of geometric models $[\![\Pi]\!]$ and $[\![\Pi']\!]$ are indenpendent if and only if the geometric model $[\![\Pi \| \Pi']\!]$ of their parallel composition is isomorphic to the cartesian product $[\![\Pi]\!] \times [\![\Pi']\!]$. Thus the decompositions of a program correspond to the factorizations if its geometric model (with respect to the cartesian product). The next subsection reminds some algebraic settings and results needed for a notion like *factorization* to make sense.

3.2 Free Commutative Monoids

The reader not familiar with this notion can refer for instance to [Lan02]. Let M be a commutative monoid. Any element of M which has an inverse is called a **unit**. A *non-unit* element x of M is said to be **irreducible** when for all y and z in M, if $x = yz$ then y or z is a unit. The set of irreducible elements of M is denoted by $I(M)$.

For any elements x and y of M, say x **divides** y when there is an element x' of M such that $xx' = y$. A *non-unit* element x of M is said to be **prime** when for all y and z in M, if x divides yz then x divides y or x divides z. The set of prime elements of M is denoted by $P(M)$.

Given a set X, the collection of maps ϕ from X to $\mathbb{N}$ such that $\{x \in X \mid \phi(x) \neq 0\}$ is finite, together with the pointwise addition, forms a commutative monoid whose neutral element is the null map: we denote it by $F(X)$. Yet, given any subset X of a commutative monoid M, the following map

$$\begin{tikzcd} \Phi_M^X : & F(X) \arrow[r] & M \\ & \phi \arrow[r, mapsto] & \prod_{x \in X} x^{\phi(x)} \end{tikzcd}$$

is a well-defined morphism of monoids. A well-known result asserts that the following are equivalent [Lan02]:

1. The mapping $\Phi_M^{I(M)}$ is an isomorphism of monoids
2. The set $I(M)$ generates[6] M and $I(M) = P(M)$
3. Any element of M can be written as a product of irreducible elements in a unique way up to permutation of terms (unique decomposition property).

[5] Categorists would write "fibre" instead of "cylinder".

[6] $X \subseteq M$ generates M when all its elements can be written as a product of elements of X. The product of the empty set being defined as the neutral element. Remark then that "$I(M)$ generates M" implies that the only unit of M is its neutral element.

If M satisfies any of the preceding assertions, then it is said to be a **free** commutative monoid. Two standard examples of free commutative monoids are given by the set of nonzero natural numbers $\mathbb{N}\backslash\{0\}$ together with multiplication (the unit is 1 and the irreducible elements are the prime numbers) and the set of natural numbers $\mathbb{N}$ together with addition (the unit is 0 and the only irreducible element is 1).

3.3 Cartesian Product and Commutation

The geometric model of a concurrent program is a set of points in an euclidean space of finite dimension. Each point is represented by the tuple of its coordinates so a geometric model is a set of tuples (whose length corresponds to the dimension of the space). The cartesian product on such structures is the following:

$$X \times Y \quad = \quad \left\{ \ (x_1, ..., x_n, y_1, ..., y_k) \ \middle| \ (x_1, ..., x_n) \in X, (y_1, ..., y_k) \in Y \ \right\}$$

However, this operator is not commutative whereas the parallel composition of programs should be so. Thus, in order to model parallel composition, we make the operator × commutative by allowing the permutation of coordinates. In the next subsection we prove a freeness theorem for a monoid generalizing this idea: tuples of (real) coordinates are replaced by words over an arbitrary (potentially infinite) alphabet. We will define a free commutative monoid of which every geometric model of a PV program is an element. From the decomposition of such models we will deduce the processes factorization.

3.4 Homogeneous Sets of Words

Let $\mathbb{A}$ be a set called the alphabet. The *noncommutative* monoid of words $\mathbb{A}^*$ consists on the finite sequences of elements of $\mathbb{A}$ together with **concatenation**. Given words w and w' of length n and n', the word $w * w'$ of length $n + n'$ is defined by

$$(w * w')_k = \begin{cases} w_k & \text{if} \qquad 1 \leqslant k \leqslant n \\ w'_{k-n} & \text{if} \quad n+1 \leqslant k \leqslant n+n' \end{cases}$$

The length of a word w is also refered to as $\ell(w)$. A **subword** of w is a word of the form $w \circ \phi$ where ϕ is a strictly increasing map $\{1, \ldots, n\} \to \{1, \ldots, \ell(w)\}$. Hence a subword of w is also entirely characterized by the image of the increasing map ϕ i.e. by a subset of $\{1, \ldots, \ell(w)\}$. If A is the image of ϕ then we write $w \circ A$ instead of $w \circ \phi$.

The n^{th} *symmetric group* $\mathfrak{S}_n$ (the group of permutations of the set $\{1, ..., n\}$) acts on the set of words of length n by composing on the right, that is for all $\sigma \in \mathfrak{S}_n$ and all words w of length n we have

$$\sigma \cdot w := w \circ \sigma = (w_{\sigma(1)} \cdots w_{\sigma(n)})$$

The concatenation extends to sets of words. Given $S, S' \subseteq \mathbb{A}^*$, define

$$S * S' := \{w * w' \mid w \in S; w' \in S'\}$$

Remark that this concatenation of sets corresponds to the cartesian product.

The set $\mathcal{P}(\mathbb{A}^*)$ of subsets of $\mathbb{A}^*$ is thus endowed with a structure of *non-commutative* monoid whose neutral element is $\{\epsilon\}$: the singleton containing the empty word. Note that the empty set $\emptyset$ is the *absorbing* element of $\mathcal{P}(\mathbb{A}^*)$, that is for all $S \subseteq \mathbb{A}^*$ we have

$$\emptyset * S = S * \emptyset = \emptyset$$

A subset S of $\mathbb{A}^*$ is said to be **homogeneous** when all the words it contains share the same length n. By analogy with the geometric construction, n is called the dimension of S and denoted by $d(S)$. The symmetric group $\mathfrak{S}_n$ acts on the set of homogeneous sets of dimension n in a natural way by applying the same permutation to all words:

$$\sigma \cdot S := \{\sigma \cdot w \mid w \in S\}$$

The homogeneous subsets of $\mathbb{A}^*$ form a sub-monoid $\mathcal{P}_h(\mathbb{A}^*)$ of $\mathcal{P}(\mathbb{A}^*)$ and can be equipped with an equivalence relation as follows: write $S \sim S'$ when $d(S) = d(S') = n$ and there exists $\sigma \in \mathfrak{S}_n$ such that $S' = \sigma \cdot S$. Moreover, for two permutations $\sigma \in \mathfrak{S}_n$ and $\sigma' \in \mathfrak{S}_{n'}$, define the **juxtaposition** $\sigma \otimes \sigma' \in \mathfrak{S}_{n+n'}$ as:

$$\sigma \otimes \sigma'(k) := \begin{cases} \sigma(k) & \text{if} \quad 1 \leqslant k \leqslant n \\ \big(\sigma'(k-n)\big) + n & \text{if} \quad n+1 \leqslant k \leqslant n+n' \end{cases}$$

A Godement-like exchange law is satisfied, which ensures that $\sim$ is actually a congruence:

$$(\sigma \cdot S) * (\sigma' \cdot S') = (\sigma \otimes \sigma') \cdot (S * S')$$

Hence the quotient $\mathcal{P}_h(\mathbb{A}^*)/{\sim}$ from which the absorbing element has been removed is still a monoid called the **homogeneous monoid** over $\mathbb{A}$ and denoted by $\mathcal{H}(\mathbb{A})$. Moreover the homogeneous monoid is commutative and its only unit is the singleton $\{\epsilon\}$. Remark that if the alphabet $\mathbb{A}$ is a singleton (resp. the empty set) then the homogeneous monoid $\mathcal{H}(\mathbb{A})$ is isomorphic to $(\mathbb{N}, +, 0)$ (resp. the null monoid). From now on the elements of $\mathcal{P}_h(\mathbb{A}^*)$ are denoted by capital letters S, S', S_k (and so on) while capital letters H, H', H_k are used to denote the elements of $\mathcal{H}(\mathbb{A})$. As they are $\sim$-equivalence classes, the elements of $\mathcal{H}(\mathbb{A})$ are subsets of $\mathcal{P}_h(\mathbb{A}^*)$. In particular for any $H \in \mathcal{H}(\mathbb{A})$ and any $S, S' \in H$ we have $d(S) = d(S')$ so we can soundly define the dimension of H as $d(H) := d(S)$.

Theorem 1. *For any set $\mathbb{A}$ the homogeneous monoid over $\mathbb{A}$ is free.*

Proof. We check the characterizing condition 2 of the Section 3.2. From the equality $d(H * H') = d(H) + d(H')$ and a straightforward induction on the dimension of elements of $\mathcal{H}(\mathbb{A})$ we deduce they can all be written as products of irreducible elements: $I(\mathcal{H}(\mathbb{A}))$ generates $\mathcal{H}(\mathbb{A})$.

Now suppose H is an irreducible element of $\mathcal{H}(\mathbb{A})$ which divides $H_1 * H_2$ and pick S, S_1 and S_2 respectively from the equivalence classes H, H_1 and H_2. Define $n = d(S)$, $n_1 = d(S_1)$ and $n_2 = d(S_2)$, and remark that $n \leqslant n_1 + n_2$. There exists $\sigma \in \mathfrak{S}_n$ and some S_3 such that $\sigma \cdot (S_1 * S_2) = S * S_3$ in $\mathcal{P}_h(\mathbb{A}^*)$. Suppose in addition that H does not divide H_1 nor H_2, then we have $A_1 \subseteq \{1, ..., n_1\}$ and $A_2 \subseteq \{1, ..., n_2\}$ s.t. $A_1 \neq \emptyset$, $A_2 \neq \emptyset$ and $\sigma(A_1 \cup A'_2) = \{1, ..., n\}$ where $A'_2 := \{a + n_1 \mid a \in A_2\}$. Then we have a nontrivial factoring $S = S'_1 * S'_2$ where

$$S'_1 := \{w \circ A_1 \mid w \in S_1\} \text{ and } S'_2 := \{w \circ A_2 \mid w \in S_2\}$$

This contradicts irreducibility of H. Hence H divides H_1 or H_2 and thus H is prime. So any irreducible element of $\mathcal{H}(\mathbb{A})$ is prime: $I(\mathcal{H}(\mathbb{A})) \subseteq P(\mathcal{H}(\mathbb{A}))$.

Finally, suppose H is a prime element of $\mathcal{H}(\mathbb{A})$ such that $H = H_1 * H_2$. In particular H divides $H_1 * H_2$, and since H is prime it divides H_1 or H_2. Both cases being symmetrical, suppose H divides H_1. In particular $d(H) \leq d(H_1)$. On the other hand $d(H) = d(H_1) + d(H_2)$, and thus $d(H_2) \leq 0$. Dimensions being natural numbers, we deduce that $d(H_2) = 0$ and then that $H_2 = \{\epsilon\}$. Hence H is irreducible, and $I(\mathcal{H}(\mathbb{A})) = P(\mathcal{H}(\mathbb{A}))$.

A useful feature of the construction is that any binary relation $\diamond$ over $\mathcal{P}_h(\mathbb{A}^*)$ which is compatible with the product and satifies

$$\forall S, S' \in \mathcal{P}_h(\mathbb{A}^*) \ \big(d(S) = d(S') \text{ and } S \diamond S' \ \Rightarrow \ \forall \sigma \in \mathfrak{S}_{d(S)} \ (\sigma \cdot S) \diamond (\sigma \cdot S')\big)$$

can be extended to a relation on $\mathcal{H}(\mathbb{A})$ which is still compatible with the product. Actually it suffices to set $H \diamond H'$ when $d(H) = d(H')$ and there exists a representative S of H and a representative S' of H' such that for all $\sigma \in \mathfrak{S}_{d(H)}$ we have $(\sigma \cdot S) \diamond (\sigma \cdot S')$. In addition, if the relation $\diamond$ satisfies

$$\forall S, S' \in \mathcal{P}_h(\mathbb{A}^*) \ S \diamond S' \ \Rightarrow \ d(S) = d(S')$$

then the quotient map is compatible with $\diamond$ and its extension. The relation of inclusion $\subseteq$ over $\mathcal{P}_h(\mathbb{A}^*)$ obviously satisfies these properties and therefore extends to $\mathcal{H}(\mathbb{A})$.

3.5 Cubical Areas

The monoid $\mathcal{P}_h(\mathbb{R}^*)$ is ordered by inclusion, according to the preceding section the relation $\subseteq$ is then extended to $\mathcal{H}(\mathbb{R})$ by setting $H \preccurlyeq H'$ when $d(H) = d(H')$ and there exist $S \in H$ and $S' \in H'$ such that for all $\sigma \in \mathfrak{S}_{d(H)}$ we have $\sigma \cdot S \subseteq \sigma \cdot S'$.

A **cube** of dimension n is a word of length n on the alphabet $\mathcal{I}$ of *nonempty* intervals of $\mathbb{R}$ so it can also be seen as a subset of $\mathbb{R}^n$. In particular, given $S \in \mathcal{P}_h(\mathcal{I}^*)$ we can define the set theoretic union

$$\bigcup_{C \in S} C$$

as a subset of $\mathbb{R}^n$ and thus an element of $\mathcal{P}_h(\mathbb{R}^*)$ provided we identify any word of length n over $\mathbb{R}$ with a point of $\mathbb{R}^n$.

The elements of $\mathcal{H}(\mathcal{I})$ are called the **cubical coverings** and we will use the capital letters F, F' or F_k ($k \in \mathbb{N}$) to denote them. Furthermore the homogeneous monoid $\mathcal{H}(\mathcal{I})$ is endowed with a preorder arising from the inclusion on $\mathcal{I}$. Indeed, given two homogeneous sets of cubes of the same dimension S and S' we write $S \preccurlyeq S'$ when for all cubes $C \in S$ there exists a cube $C' \in S'$ such that $C \subseteq C'$. The relation $\preccurlyeq$ provides the monoid $\mathcal{P}_h(\mathcal{I}^*)$ with a preorder that can be extended to $\mathcal{H}(\mathcal{I})$ by setting $F \preccurlyeq F'$ when $d(F) = d(F')$ and there exist $S \in F$ and $S' \in F'$ such that for all $\sigma \in \mathfrak{S}_{d(F)}$ we have $\sigma \cdot S \preccurlyeq \sigma \cdot S'$. We now establish a Galois connection between $(\mathcal{H}(\mathbb{R}), \preccurlyeq)$ and $(\mathcal{H}(\mathcal{I}), \preccurlyeq)$. Given a cubical covering F one can check that the following is actually an element of $\mathcal{H}(\mathbb{R})$.

$$\gamma(F) := \Big\{ \bigcup_{C \in S} C \;\Big|\; S \in F \Big\}$$

The mapping γ is a morphism of monoids and if $F \preccurlyeq F'$ then $\gamma(F) \preccurlyeq \gamma(F')$. Conversely, given some $S \in \mathcal{P}_h(\mathbb{R}^*)$ the collection of n-dimensional cubes C such that $C \subseteq S$, ordered by inclusion, is a semilattice whose maximal elements are called the **maximal cubes** of S. The set M_S of maximal cubes of S is homogeneous and for all $\sigma \in \mathfrak{S}_n,\;\; \sigma \cdot M_S = M_{\sigma \cdot S}$. Then given $H \in \mathcal{H}(\mathbb{R})$ one can check that the following is actually an element of $\mathcal{H}(\mathcal{I})$.

$$\alpha(H) := \Big\{ M_S \;\Big|\; S \in H \Big\}$$

Furthermore α is a morphism of monoids and if $H \preccurlyeq H'$ then $\alpha(H) \preccurlyeq \alpha(H')$. Then we have a Galois connection:

Proposition 1. $\gamma \circ \alpha = \mathsf{id}_{\mathcal{H}(\mathbb{R})}$ *and* $\mathsf{id}_{\mathcal{H}(\mathcal{I})} \preccurlyeq \alpha \circ \gamma$.

Given $H \in \mathcal{H}(\mathbb{R})$ and $F \in \mathcal{H}(\mathcal{I})$ we say that F is a cubical covering of H when $\gamma(F) = H$. The **cubical areas** are the elements H of $\mathcal{H}(\mathbb{R})$ which admit a *finite* cubical covering. The collection of cubical areas (resp. finite cubical coverings) forms the submonoid Are of $\mathcal{H}(\mathbb{R})$ (resp. Cov of $\mathcal{H}(\mathcal{I})$). The restrictions of the morphisms γ and α to Cov and Are induce another Galois connection.

Proposition 2. $\gamma \circ \alpha = \mathsf{id}_{\mathtt{Are}}$ *and* $\mathsf{id}_{\mathtt{Cov}} \preccurlyeq \alpha \circ \gamma$.

Moreover, the morphisms γ and α of Proposition 2 induce a pair of isomorphisms of commutative monoids between Are and the collection of fixpoints of $\alpha \circ \gamma$. A submonoid of a free commutative monoid may not be free. Yet, under a simple additional hypothesis this pathological behaviour is no more possible. We say that a submonoid P of a monoid M is **pure** when for all $x, y \in M$, $x * y \in P \;\Rightarrow\; x \in P$ and $y \in P$.

Lemma 1. *Every pure submonoid of a free commutative monoid is free.*

Proof. Let P be a pure submonoid of a free commutative monoid M. Let p be an element of P written as a product $x_1 \cdots x_n$ of irreducible elements of M. Each

x_i is obviously an irreducible element of P so any element of P can be written as a product of irreducible elements of P. Furthermore any irreducible element of P is also an irreducible element of M because P is pure in M. It follows that any elements of P can be written as a product of irreducible elements of P in a unique way i.e. P is free. Then we have:

Theorem 2. *The commutative monoid of cubical areas is free.*

Proof. Let X and X' be two elements of $\mathcal{H}(\mathbb{R})$ and suppose $X * X'$ belongs to $\texttt{Are}$. Since both α and γ are morphisms of monoids we have $\alpha \circ \gamma(X * X') = \alpha \circ \gamma(X) * \alpha \circ \gamma(X')$ which is finite. It follows that both $\alpha \circ \gamma(X)$ and $\alpha \circ \gamma(X')$ are finite. Hence X and X' actually belongs to $\texttt{Are}$, which is thus free as a pure submonoid of $\mathcal{H}(\mathbb{R})$.

Moreover one can check that for any $n \in \mathbb{N}$ and any finite family $C_1, \dots, C_k$ of bounded[7] n-dimensional cubes, $\mathbb{R}^n \backslash (C_1 \cup \dots \cup C_k)$ is irreducible. Therefore the commutative monoid of cubical areas has infinitely many irreducible elements.

The Theorem 2 is the theoretical cornerstone of our method: the geometric model of a PV program is an element of $\mathcal{H}(\mathcal{I})$ so we obtain from its decomposition the expected processes factorization.

4 Effective Factoring of Cubical Areas

Beyond their theoretical usefulness, the maximal cubes provide the data structure which allows to handle algorithmically cubical areas, as in the static analyzer ALCOOL which is devoted to the study of parallel programs.

4.1 Implementation

We need an algorithm which performs decompositions in $\mathcal{H}(\mathbb{A})$, its implementation is directly based on the proof of Theorem 1: $H \in \mathcal{H}(\mathbb{A})$ is reducible if and only if there exists some representative S of H which admits a nontrivial decomposition in $\mathcal{P}_h(\mathbb{A}^*)$. In order to describe the algorithm we define

$$S \circ A := \{w \circ A \mid w \in S\}$$

for any $S \in \mathcal{P}_h(\mathbb{A}^*)$ and $A \subseteq \{1, ..., d(S)\}$. Moreover for $w' \in \mathbb{A}^*$ with $\ell(w') = |A|$ and A^c the complement of A (in $\{1, ..., d(S)\}$), we define the set of words

$$\Psi(w', A, S) := \{w \circ A^c \mid w \in S \text{ and } w \circ A = w'\}$$

Then the class $[S \circ A] \in \mathcal{H}(\mathbb{A})$ divides H if and only if for all $w' \in S \circ A$ one has $\Psi(w', A, S) = [S \circ A^c]$. In particular the choice of $S \in H$ does not alter the result of the test and we have

$$[S \circ A] * [S \circ A^c] = H$$

[7] An n-dimensional cube C is bounded when $C \subseteq [-r, r]^n$ for some $r > 0$.

Then we look for some divisor of H by testing all the nonempty subsets A of $\{1, \ldots, d(S)\}$ (each test requires that we look over all the elements of $S \circ A$) according to the following total ordering

$$A \leqslant A' \text{ when } |A| < |A'| \text{ or } (|A| = |A'| \text{ and } A \sqsubseteq_{\text{lex}} A')$$

where $\sqsubseteq_{\text{lex}}$ is the lexicographic ordering ($A \sqsubseteq_{\text{lex}} A'$ is recursively defined by $\min(A) < \min(A')$ or ($\min(A) = \min(A')$ and $A\backslash\{\min(A)\} \sqsubseteq_{\text{lex}} A'\backslash\{\min(A')\}$). Doing so, we know that if A is the first value such that $[S \circ A]$ divides H, then $[S \circ A]$ is irreducible. Moreover we have $d([S \circ A]) = |A|$ and for all $H_0, H_1 \in \mathcal{H}(\mathbb{A})$, $d(H_0 * H_1) = d(H_0) + d(H_1)$ hence we can suppose

$$|A| \leqslant \frac{d(H)}{2} + 1$$

The software `ALCOOL` is entirely written in `OCaml`. The complexity of the decomposition algorithm implemented in it is exponential in the dimension n of the cubical area since it checks all the subsets of $\{0, \ldots, n-1\}$. However, it is worth remarking that our algorithm is efficient when the cubical area to decompose is actually the product of several irreducible cubical areas of small dimension (see Subsection 4.2 for benchmarks). This remark should be compared with the fact that the standard decomposition algorithm of integer into primes is very efficient on products of small prime numbers.

We treat the case of the program Σ given in Example 1. Denote by H its geometric model, we are actually provided with some representative $S \in H$. With the preceding notation we then check that $[S \circ A]$ divides H for $A := \{1, 3\}$. Applying the permutation $(2, 3)$ we have

$$(2, 3) \cdot \{\{1, 3\}, \{2, 4\}\} = \{\{1, 2\}, \{3, 4\}\}$$

then $(2, 3) \cdot S$ can be decomposed in $\mathcal{P}_h(\mathbb{R}^*)$ as

$$\Big(\texttt{[0,1[*[0,-[} \parallel \texttt{[4,-[*[0,-[} \parallel \texttt{[0,-[*[0,1[} \parallel \texttt{[0,-[*[4,-[}\Big)^2$$

and it follows that in the program Σ the sets of processes $\{\pi_1, \pi_3\}$ and $\{\pi_2, \pi_4\}$ run independently from each other.

4.2 Benchmarks

We describe some programs upon which the algorithm has been tested. The program $\Sigma_{n_1,\ldots,n_k}$ is made of k groups of processes: for all $i \in \{1, ..., k\}$ it contains n_i copies of the process

$$P(a_i).P(b).V(b).V(a_i)$$

where a_i is a mutex and b is a semaphore of arity $k + 1$. All processes then share the resource b, but as for Σ in Example 1 the k groups are actually independent. On the other hand the program $\Sigma'_{n_1,\ldots,n_k}$ is the same as $\Sigma_{n_1,\ldots,n_k}$ but with b of arity only k, which forbids any decomposition. The n-philosophers programs implement the standard n dining philosophers algorithm. The benchmark table of Figure 3 has been obtained using the Unix command `time` which is not accurate. Hence these results have to be understood as an overapproximation of the mean execution time.

Example	Time (in sec.)	Decomp.
6 philosophers	0.2	No
7 philosophers	0.7	No
8 philosophers	3.5	No
9 philosophers	21	No
10 philosophers	152	No

Example	Time (in sec.)	Decomp.	Example	Time (in sec.)	Decomp.
$\Sigma_{2,2}$	0.1	$\{1,3\}\{2,4\}$	$\Sigma'_{2,2}$	0.1	No
$\Sigma_{2,2,2}$	0.1	$\{1,4\}\{2,5\}\{3,6\}$	$\Sigma'_{2,2,2}$	0.3	No
$\Sigma_{3,3}$	0.13	$\{1,3,5\}\{2,4,6\}$	$\Sigma'_{3,3}$	0.52	No
$\Sigma_{2,2,2,2}$	0.13	$\{1,5\}\{2,6\}\{3,7\}\{4,8\}$	$\Sigma'_{2,2,2,2}$	7.1	No
$\Sigma_{4,4}$	1	$\{1,3,5,7\}\{2,4,6,8\}$	$\Sigma'_{4,4}$	33	No
$\Sigma_{3,3,3}$	1.5	$\{1,4,7\}\{2,5,8\}\{3,6,9\}$	$\Sigma'_{3,3,3}$	293	No
$\Sigma_{4,5}$	6.1	$\{1,3,5,7\}\{2,4,6,8\}$	$\Sigma'_{4,5}$	327	No
$\Sigma_{5,5}$	50	$\{1,3,5,7,9\}\{2,4,6,8,10\}$	$\Sigma'_{5,5}$	2875	No

Fig. 3. Benchmarks

5 Conclusion

Related work

The problem of decomposition of concurrent programs in *CCS*-style has been studied in [GM92] and [MM93]. By the possibility of using semaphores of arbitrary arity, our work seems to go beyond this previous approach. Also note that the silent and synchronous communication mechanism of *CCS* can be given a straightforward geometric interpretation which falls in the scope of the present discussion. However, the link between bisimilarity in *CCS* and isomorphic geometric interpretations is still to be explored to make clear the relations between these works.

In [LvO05] *B. Luttik* and *V. van Oostrom* have characterized the commutative monoids with unique decomposition property as those which can be provided with a so-called decomposition order. In the case where the property holds, the divisibility order always fits. Yet, there might exist a more convenient one. Unfortunately, in the current setting the authors are not aware of any such order yielding direct proofs. Nevertheless it is worth noticing that this approach is actually applied for decomposition of processes in a normed ACP theory for which a convenient decomposition order exists.

One can also think of using this method to optimize the implementation of parallel programs. In the same stream of ideas, [CGR97] defines a preorder over a simple process algebra with durational actions in order to compare the implementations of a same algorithm according to their efficiency.

Conclusion

This paper uses a geometric semantics for concurrent programs, and presents a proof of a unique decomposition property together with an algorithm working at this semantic level (Theorem 2). The main strength of this work is that it applies to any concurrent program yielding a cubical area. Example of features allowed in this setting are: semaphores, synchronisation barriers, synchronous as well as asynchronous communications (with finite or infinite message queues), conditional branchings. In fact we can even consider loops provided we replace the set $\mathcal{I}$ of intervals of the real line $\mathbb{R}$ by the set $\mathcal{A}$ of arcs of the circle.

Future work

Any cubical area naturally enjoys a pospace[8] structure. Pospaces are among the simplest objects studied in *Directed Algebraic Topology*. In particular, a cubical area is associated with its category of components [FGHR04, GH05, Hau06] and [GH07], which is proven to be finite, loop-free[9] and in most cases connected. Then, as the cubical areas do, these categories together with cartesian product form a free commutative monoid. It is worth noticing this is actually the generalization of a result concerning finite posets which has been established in the early fifties [Has51]. Therefore a program Π can be decomposed by lifting the decomposition of the category of components of its geometric model $[\![\Pi]\!]$. In general, the relation between the decomposition of a cubical area and the one of its category of components is a theoretical issue the authors wish to investigate.

Another important concern is a clarification of the control constructs compatible with cubical areas: replacing in some dimensions the intervals of the real line by the arcs of the circle as mentioned above corresponds to a global loop, but some richer structures may be useful.

A final point of interest is the investigation of the exact relation between our decomposition results and the ones of [GM92, MM93, LvO05]. Indeed they use *CCS*-like syntaxes to describe some classes of edge-labelled graphs modulo bisimilarity, whereas the category of components of our models correspond to some other graphs modulo directed homotopy. Hence the question: what is in this setting the relation between bisimilarity and directed homotopy?

References

[BMP99] Bournez, O., Maler, O., Pnueli, A.: Orthogonal polyhedra: Representation and computation. In: Vaandrager, F.W., van Schuppen, J.H. (eds.) HSCC 1999. LNCS, vol. 1569, p. 46. Springer, Heidelberg (1999)

[CR87] Carson, S.D., Reynolds Jr., P.F.: The geometry of semaphore programs. ACM Transactions on Programming Languages and Systems, 25–53 (1987)

[8] Shorthand for "partially ordered spaces" [Nac65].

[9] Loop-free categories were introduced in [Hae91, Hae92] as "small categories without loop" or "scwols" in a context far away from ours.

[CGR97] Corradini, F., Gorrieri, R., Roccetti, R.: Performance Preorder and Competitive Equivalence. Acta Inf. 34, 805–835 (1997)

[Dij68] Dijkstra, E.W.: Cooperating sequential processes. In: Programming Languages: NATO Advanced Study Institute, pp. 43–112. Academic Press, London (1968)

[FGHR04] Fajstrup, L., Goubault, E., Haucourt, E., Raußen, M.: Component categories and the fundamental category. App. Cat. Struct. 12, 81–108 (2004)

[GH05] Goubault, E., Haucourt, E.: A practical application of geometric semantics to static analysis of concurrent programs. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 503–517. Springer, Heidelberg (2005)

[GH07] Goubault, E., Haucourt, E.: Component categories and the fundamental category II. App. Cat. Struct. 15 (2007)

[GM92] Groote, J.F., Moller, F.: Verification of Parallel Systems via Decomposition. In: Cleaveland, W.R. (ed.) CONCUR 1992. LNCS, vol. 630, pp. 62–76. Springer, Heidelberg (1992)

[Hae91] Haefliger, A.: Complexes of groups and orbihedra. In: Group theory from a geometrical viewpoint, pp. 504–540. World Scientific, Singapore (1991)

[Hae92] Haefliger, A.: Extension of complexes of groups. Annales de l'institut Fourier 42(1-2), 275–311 (1992), http://www.numdam.org/

[Has51] Hashimoto, J.: On direct product decomposition of partially ordered sets. Annals of Mathematics 54, 315–318 (1951)

[Hau06] Haucourt, E.: Categories of components and Loop-free categories. Theory and Applications of Categories 16(27), 736–770 (2006)

[Lan02] Lang, S.: Algebra. Springer, Heidelberg (2002) (4th corrected printing)

[LvO05] Luttik, B., van Oostrom, V.: Decomposition orders: another generalisation of the fundamental theorem of arithmetic. Theor. Comp. Sci. 335, 147–186 (2005)

[MM93] Milner, R., Moller, F.: Unique Decomposition of Processes. Theor. Comp. Sci. 107, 357–363 (1993)

[Nac65] Nachbin, L.: Topology and Order. Van Nostrand Mathematical Studies, vol. 4. Van Nostrand, Princeton (1965)

[Tha09] Thao, D.: Methods and Tools for Computer Aided Design of Embedded Systems. HDR Thesis, ch. 5 (2009)

[Win95] Winskel, G.: Handbook of Logic in Computer Science. Semantic Modelling, vol. 4. ch. 1. Oxford University Press, Oxford (1995)

A Logic for True Concurrency*

Paolo Baldan and Silvia Crafa

Department of Pure and Applied Math, University of Padova

Abstract. We propose a logic for true concurrency whose formulae predicate about events in computations and their causal dependencies. The induced logical equivalence is hereditary history preserving bisimilarity, and fragments of the logic can be identified which correspond to other true concurrent behavioural equivalences in the literature: step, pomset and history preserving bisimilarity. Standard Hennessy-Milner logic, thus (interleaving) bisimilarity, is also recovered as a fragment. We believe that this contributes to a rational presentation of the true concurrent spectrum and to a deeper understanding of the relations between the involved behavioural equivalences.

1 Introduction

In the semantics of concurrent and distributed systems, a major dichotomy opposes the *interleaving approaches*, where concurrency of actions is reduced to the non-deterministic choice among their possible sequentializations, to *true-concurrent approaches*, where concurrency is taken as a primitive notion. In both cases, on top of the operational models a number of behavioural equivalences have been defined by abstracting from aspects which are considered unobservable [1,2].

For the interleaving world, a systematic and impressive picture is taken in the linear-time branching-time spectrum [1]. Quite interestingly, the equivalences in the spectrum can be uniformly characterised in logical terms. Bisimilarity, the finest equivalence, corresponds to Hennessy-Milner (HM) logic: two processes are bisimilar if and only if they satisfy the same HM logic formulae [3]. Coarser equivalences correspond to suitable fragments of HM logic.

In the true-concurrent world, relying on models like event structures or transition systems with independence [4], several behavioural equivalences have been defined, ranging from hereditary history preserving (hhp-) bisimilarity, to pomset and step bisimilarity. Correspondingly, a number of logics have been studied, but, to the best of our knowledge, a unifying logical framework encompassing the main true-concurrent equivalences is still missing. The huge amount of work on the topic makes it impossible to give a complete account of related approaches. Just to give a few references (see Section 5 for a wider discussion), [5] proposes a general framework encompassing a number of temporal and modal

* Supported by the MIUR Projects `SisteR` and `AIDA2007`, and the project `AVIAMO` of the University of Padova.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 147–161, 2010.

logics that characterize pomset and weak history preserving bisimilarities as well as interleaving bisimilarity. However, finer equivalences are not considered and a single unitary logic is missing. History preserving (hp-) bisimilarity has been characterised in automata-theoretic terms using HD-automata [6] or Petri nets [7]. More recently, hp-bisimilarity has been obtained as a logical equivalence, using a separation fixpoint modal logic where it is possible to check the causal dependence/independence of the last executed action [8,9]. Concerning hhp-bisimilarity, several logics with modalities corresponding to the "retraction" or "backward" execution of computations have been proposed [10,11,12]. In absence of autoconcurrency they are shown to capture hhp-bisimilarity, while the general case complicates the picture and requires some adjustments.

In this paper we propose a behavioural logic for concurrency and we show that it allows to characterise a relevant part of the truly concurrent spectrum. More specifically, the full logic is shown to capture hhp-bisimilarity, the finest equivalence in the spectrum [2]. Then suitable fragments of the logic are shown to scale down to the characterisation of other coarser equivalences, i.e., history preserving, pomset and step bisimilarity. Standard HM logic, and thus (interleaving) bisimilarity, is also recovered as a fragment.

Our logic allows to predicate about events in computations together with their causal and independence relations. It is interpreted over prime event structures, but it could naturally be interpreted over any formalism with a notion of event, causality and consistency. A formula is evaluated in a configuration representing the current state of the computation, and it predicates on a set of possible future evolutions starting from that state. The logic is event-based in the sense that it contains a binder that allows to refer later to the events the formula predicates on. In this respect, it is reminiscent of the modal analogue of independence-friendly modal logic as considered in [13].

The logic contains two main operators. The formula $(x, \overline{y} < \mathsf{a}\, z)\varphi$ declares that an a-labelled future event exists, which causally depends on the event bound to x, and is independent from the event bound to y. Such an event is bound to variable z so that it can be later referred to in φ. In general, x and y can be replaced by lists of variables. A second operator allows to "execute" events previously bound to variables. The formula $\langle z \rangle\, \varphi$ says that the event bound to z is enabled in the current state, and after its execution φ holds.

Different behavioural equivalences are induced by fragments of the logics where we suitably restrict the set of possible futures the formulae are able to refer to. Namely, hhp-bisimilarity, that is captured by the full logic, corresponds to the ability of observing the existence of a number of legal but (possibly) incompatible futures, without executing such futures. Interestingly, the definition of hhp-bisimilarity is normally given in terms of backward transitions, whereas our logical characterization has a "forward flavor". By restricting to a fragment where future events can be observed only by executing them (any occurrence of the binding operator is immediately followed by a corresponding execution), we get hp-bisimilarity. Pomset bisimilarity is induced by a fragment of the logic obtained by further restricting that for hp-bisimilarity, with the requirement that propositional connectives are used only on closed (sub)formulae. Roughly speaking, this fragment predicates about the possibility of executing pomset transitions and the closedness requirement prevents pomset transitions from being causally linked to the events in the past. Finally, quite intuitively, step bisimilarity corresponds to the possibility of observing only currently enabled concurrent actions.

We believe that this work contributes to the definition of a logical counterpart of the true concurrent spectrum and shades further light on the relations between the involved behavioural equivalences.

2 Background

In this section we provide the basics of prime event structures which will be used as models for our logic. Then we define some common behavioural concurrent equivalences which will play a basic role in the paper.

2.1 Event Structures

Prime event structures [14] are a widely known model of concurrency. They describe the behaviour of a system in terms of events and dependency relations between such events. Throughout the paper Λ denotes a fixed set of labels ranged over by $\mathsf{a}, \mathsf{b}, \mathsf{c} \ldots$

Definition 1 (prime event structure). *A (Λ-labelled)* prime event structure *(*PES*) is a tuple $\mathcal{E} = \langle E, \leq, \#, \lambda \rangle$, where E is a set of* events, *$\lambda : E \to \Lambda$ is a labelling function and $\leq$, $\#$ are binary relations on E, called* causality *and* conflict *respectively, such that:*

1. *$\leq$ is a partial order and $\lceil e \rceil = \{e' \in E \mid e' \leq e\}$ is finite for all $e \in E$;*
2. *$\#$ is irreflexive, symmetric and hereditary with respect to $\leq$, i.e., for all $e, e', e'' \in E$, if $e \# e' \leq e''$ then $e \# e''$.*

In the following, we will assume that the components of an event structure $\mathcal{E}$ are named as in the definition above. Subscripts carry over to the components.

Definition 2 (consistency, concurrency). *Let $\mathcal{E}$ be a* PES*. We say that $e, e' \in E$ are* consistent*, written $e \frown e'$, if $\neg(e \# e')$. Moreover, we say that e and e' are* concurrent*, written $e \,||\, e'$, if $\neg(e \leq e')$, $\neg(e' \leq e)$ and $\neg(e \# e')$.*

Causality and concurrency will be sometimes used on set of events. Given $X \subseteq E$ and $e \in E$, by $X < e$ we mean that for all $e' \in X$, $e' < e$. Similarly $X \,||\, e$, resp. $X \frown e$, means that for all $e' \in X$, $e' \,||\, e$, resp. $e' \frown e$. We write $\lceil X \rceil$ for $\bigcup_{e \in X} \lceil e \rceil$.

The idea of (concurrent) computation is captured, in event structures, by the notion of configuration.

Definition 3 (configuration). *Let $\mathcal{E}$ be a* PES*. A* (finite) configuration *in $\mathcal{E}$ is a (finite) subset of events $C \subseteq E$ pairwise consistent and closed w.r.t. causality (i.e., $\lceil C \rceil = C$). The set of finite configurations of $\mathcal{E}$ is denoted by $\mathcal{C}(\mathcal{E})$.*

Hereafter, unless explicitly stated otherwise, all configurations will be assumed to be finite. A pairwise consistent subset $X \subseteq E$ of events will be always seen as the *pomset* (partially ordered multiset) $(X, \leq_X, \lambda_X)$, where $\leq_X$ and λ_X are the restrictions of $\leq$ and λ to X. Given $X, Y \subseteq E$ we will write $X \sim Y$ if X and Y are isomorphic as pomsets.

Definition 4 (pomset transition and step). *Let $\mathcal{E}$ be a* PES *and let $C \in \mathcal{C}(\mathcal{E})$. Given $\emptyset \neq X \subseteq E$, if $C \cap X = \emptyset$ and $C' = C \cup X \in \mathcal{C}(\mathcal{E})$ we write $C \xrightarrow{X} C'$ and call it a* pomset transition *from C to C'. When the events in X are pairwise concurrent, we say that $C \xrightarrow{X} C'$ is a* step. *When $X = \{e\}$ we write $C \xrightarrow{e} C'$ instead of $C \xrightarrow{\{e\}} C'$.*

A PES $\mathcal{E}$ is called *image finite* if for any $C \in \mathcal{C}(\mathcal{E})$ and $\mathsf{a} \in \Lambda$, the set of events $\{e \in E \mid C \xrightarrow{e} C' \ \wedge \ \lambda(e) = \mathsf{a}\}$ is finite. *All* the PESs considered in this paper will be assumed to be image finite. As it commonly happens when relating modal logics and bisimilarities, this assumption is crucial for getting the logical characterisation of the various bisimulation equivalences in Section 4.

2.2 Concurrent Behavioural Equivalences

Behavioural equivalences which capture to some extent the concurrency features of a system, can be defined on the transition system where states are configurations and transitions are pomset transitions.

Definition 5 (pomset, step bisimulation). *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be* PES*s. A* pomset bisimulation *is a relation $R \subseteq \mathcal{C}(\mathcal{E}_1) \times \mathcal{C}(\mathcal{E}_2)$ such that if $(C_1, C_2) \in R$ and $C_1 \xrightarrow{X_1} C_1'$ then $C_2 \xrightarrow{X_2} C_2'$, with $X_1 \sim X_2$ and $(C_1', C_2') \in R$, and vice versa. We say that $\mathcal{E}_1$, $\mathcal{E}_2$ are* pomset bisimilar*, written $\mathcal{E}_1 \sim_p \mathcal{E}_2$, if there exists a pomset bisimulation R such that $(\emptyset, \emptyset) \in R$.*

Step bisimulation *is defined analogously, replacing general pomset transitions with steps. We write $\mathcal{E}_1 \sim_s \mathcal{E}_2$ when $\mathcal{E}_1$ and $\mathcal{E}_2$ are step bisimilar.*

While pomset and step bisimilarity only consider the causal structure of the current step, (hereditary) history preserving bisimilarities are sensible to the way in which the executed events depend on events in the past. In order to define history preserving bisimilarities the following definition is helpful.

Definition 6 (posetal product). *Given two* PES*s $\mathcal{E}_1$, $\mathcal{E}_2$, the* posetal product *of their configurations, denoted $\mathcal{C}(\mathcal{E}_1) \bar{\times} \mathcal{C}(\mathcal{E}_2)$, is defined as*

$$\{(C_1, f, C_2) : C_1 \in \mathcal{C}(\mathcal{E}_1),\ C_2 \in \mathcal{C}(\mathcal{E}_2),\ f : C_1 \to C_2 \text{ isomorphism}\}$$

A subset $R \subseteq \mathcal{C}(\mathcal{E}_1) \bar{\times} \mathcal{C}(\mathcal{E}_2)$ is called a posetal relation. *We say that R is downward closed when for any $(C_1, f, C_2), (C_1', f', C_2') \in \mathcal{C}(\mathcal{E}_1) \bar{\times} \mathcal{C}(\mathcal{E}_2)$, if $(C_1, f, C_2) \subseteq (C_1', f', C_2')$ pointwise and $(C_1', f', C_2') \in R$, then $(C_1, f, C_2) \in R$.*

Definition 7 ((hereditary) history preserving bisimulation). *A* history preserving (hp-)bisimulation *is a posetal relation $R \subseteq \mathcal{C}(\mathcal{E}_1)\bar{\times}\mathcal{C}(\mathcal{E}_2)$ such that if $(C_1, f, C_2) \in R$ and $C \xrightarrow{e_1} C_1'$ then $C_2 \xrightarrow{e_2} C_2'$, with $(C_1', f[e_1 \mapsto e_2], C_2') \in R$, and vice versa. We say that $\mathcal{E}_1$, $\mathcal{E}_2$ are* history preserving (hp-)bisimilar *and write $\mathcal{E}_1 \sim_{hp} \mathcal{E}_2$ if there exists an hp-bisimulation R such that $(\emptyset, \emptyset, \emptyset) \in R$.*

A hereditary history preserving (hhp-)bisimulation *is a downward closed hp-bisimulation. The fact that $\mathcal{E}_1, \mathcal{E}_2$ are* hereditary history preserving (hhp-)bisimilar *is denoted $\mathcal{E}_1 \sim_{hhp} \mathcal{E}_2$.*

3 A Logic for True Concurrency

In this section we introduce the syntax and the semantics of our logic, where formulae predicate about events in computations and their dependencies as primitive concepts. The logic is interpreted over PESs, but it could be equivalently interpreted over any formalism with a notion of event, causality and consistency.

As a first step we define a quite general syntax and the semantics of the two distinctive operators. Our logic for concurrency will then be obtained by filtering out formulae which do not satisfy a suitable well-formedness condition.

In order to keep the notation simple, lists of variables like $x_1, \ldots, x_n$ will be denoted by $\boldsymbol{x}$ and, abusing the notation, lists will be sometimes used as sets.

Definition 8 (syntax). *Let Var be a denumerable set of variables ranged over by $x, y, z, \ldots$. The syntax of the formulae over the set of labels Λ is defined as follows:*

$$\varphi \ ::= \ (\boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\, z)\, \varphi \ \mid \ \langle z \rangle\, \varphi \mid \ \varphi \wedge \varphi \ \mid \ \neg \varphi \ \mid \ \top$$

The operator $(\boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\, z)$ acts as a binder for variable z, as clarified by the following notion of free variables in a formula.

Definition 9 (free variables). *The set of free variables of a formula φ is denoted $fv(\varphi)$ and it is inductively defined by:*

$$fv((\boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\, z)\, \varphi) = (fv(\varphi) \setminus \{z\}) \cup \boldsymbol{x} \cup \boldsymbol{y} \qquad fv(\langle z \rangle\, \varphi) = fv(\varphi) \cup \{z\}$$

$$fv(\varphi_1 \wedge \varphi_2) = fv(\varphi_1) \cup fv(\varphi_2) \qquad fv(\top) = \emptyset \qquad fv(\neg \varphi) = fv(\varphi)$$

The satisfaction of a formula is defined with respect to a configuration, representing the state of the computation. Moreover a partial function $\eta : Var \to E$ is fixed, called an *environment*, in order to bind free variables in φ to events.

Definition 10 (semantics). *Let $\mathcal{E} = \langle E, \leq, \#, \lambda \rangle$ be a PES. For $C \in \mathcal{C}(\mathcal{E})$ a configuration, φ a formula and $\eta : Var \to E$ an environment such that $fv(\varphi) \subseteq dom(\eta)$, satisfaction $\mathcal{E}, C \models_\eta \varphi$ is defined as follows:*

$\mathcal{E}, C \models_\eta (\boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\, z)\, \varphi$ *if there is $e \in E \setminus C$, such that*
- *$\lambda(e) = \mathsf{a}$ and $C \frown e$, $\eta(\boldsymbol{x}) < e$, $\eta(\boldsymbol{y}) \mid\mid e$*
- *$\mathcal{E}, C \models_{\eta'} \varphi$, where $\eta' = \eta[z \mapsto e]$*

$\mathcal{E}, C \models_\eta \langle z \rangle\, \varphi$ *if $C \xrightarrow{\eta(z)} C'$ and $\mathcal{E}, C' \models_\eta \varphi$*

The semantics of propositional connectives is defined as usual. We say that a PES $\mathcal{E}$ *satisfies a closed formula* φ*, written* $\mathcal{E} \models \varphi$*, when* $\mathcal{E}, \emptyset \models_\emptyset \varphi$*.*

Intuitively, the formula

$$(x_1 \ldots x_n, \overline{y_1 \ldots y_m} < \mathsf{a}\, z)\, \varphi$$

holds in C if in the future of such a configuration there is an a-labelled event e, and binding e to the variable z, the formula φ holds. Such an event is required to be caused (at least) by the events already bound to $x_1 \ldots x_n$, and to be independent (at least) from those bound to $y_1 \ldots y_m$. We stress that the event e might not be currently enabled; it is only required to be consistent with the current configuration, meaning that it could be enabled in the future of the current configuration. The formula $\langle z \rangle\, \varphi$ says that the event bound to z is currently enabled, hence it can be executed producing a new configuration which satisfies formula φ. To simplify the notation we write $(\mathsf{a}\, z)\, \varphi$ for $(\;\; < \mathsf{a}\, z)\, \varphi$ and we often omit trailing $\top$ at the end of the formulae.

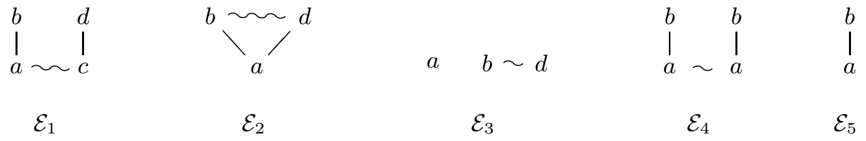


Fig. 1.

As an example, consider the PES $\mathcal{E}_1$ in Fig. 1, corresponding to the CCS process $a.b + c.d$, where curly lines represent immediate conflict and the causal order proceeds upwards along the straight lines. The empty configuration satisfies the closed formula $(\mathsf{b}\, x)$, i.e., $\mathcal{E}_1 \models (\mathsf{b}\, x)$, even if the b-labelled event is not immediately enabled. Also $\mathcal{E}_1 \models (\mathsf{b}\, x) \wedge (\mathsf{d}\, y)$, since there are two possible (incompatible) computations that starts from the empty configuration and contain, respectively, a b-labelled and a d-labelled event. On the other hand, if $\varphi = (\mathsf{a}\, z)\langle z \rangle\, ((\mathsf{b}\, x) \wedge (\mathsf{d}\, y))$ then $\mathcal{E}_1 \not\models \varphi$ since after the execution of the a-labelled event, $\mathcal{E}_1$ reaches a configuration that does not admit a future containing an event labelled by d. As a further example, the formula φ above is satisfied by the PESs $\mathcal{E}_2$ and $\mathcal{E}_3$ in Fig. 1 corresponding respectively to the process $a.(b + d)$ and $a \mid (b + d)$, whereas the formula $(\mathsf{a}\, z)\langle z \rangle\, (\overline{z} < \mathsf{b}\, x)$ is satisfied only by $\mathcal{E}_3$.

It is worth noticing that the semantics of the binding operator does not prevent from choosing for z an event e that has been already bound to a different variable, i.e., the environment function η needs not to be injective. This is essential to avoid the direct observation of conflicts. Consider for instance the PESs associated to the hhp-equivalent processes $a + a$ and a: in order to be also logically equivalent, they both must satisfy the formula $(\mathsf{a}\, z)(\mathsf{a}\, z')$. Hence for the second PES, both z and z' should be bound to the unique a-labelled event. On the other hand, observe that both PESs falsify the formula $(\mathsf{a}\,z)(\mathsf{a}\,z')\langle z\rangle\,\langle z'\rangle$, since either z and z' will be bound to the same event, which cannot be executed twice, or they will be bound to conflicting events.

Still, the logic as it is defined up to now is too powerful since it allows to observe conflicts through a combination of the binder and the execution modality. For instance, consider the PESs $\mathcal{E}_4$ and $\mathcal{E}_5$ in Fig. 1, corresponding to the processes $a.b + a.b$ and $a.b$ and take formula $\varphi = (\mathsf{a}\ x)(\mathsf{b}\ y)\langle x\rangle\,\neg\langle y\rangle$. Then $\mathcal{E}_5 \not\models \varphi$, while $\mathcal{E}_4 \models \varphi$, since only in $\mathcal{E}_4$ the variables x and y can be bound to conflicting events (e.g., x could be bound to the a-labelled event on the left and y to the b-labelled event on the right). In a similar way, the logic allows one to distinguish the PESs corresponding to any process from that corresponding to the non-deterministic choice between that process and itself, which instead are equated by virtually any behavioural equivalence.

In order to disallow the observation of conflicts and avoid this phenomenon, we restrict our logic to well-formed formulae, that "syntactically" ensure that (i) the free variables in any subformula will always refer to events consistent with the current configuration and (ii) the variables which are used as causes/non-causes, i.e., $\boldsymbol{x}$ and $\boldsymbol{y}$ in $(\boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\,z)\,\varphi$, will be bound to be pairwise consistent events.

This is formalised by the definition below. Consistency constraints are represented by a relation on variables $Co \subseteq Var \times Var$, where $(x, y) \in Co$ means that x and y must be bound to consistent events. We write $(\boldsymbol{x}, \boldsymbol{y})$ for the set $\{(x, y) : x \in \boldsymbol{x}\ \wedge\ y \in \boldsymbol{y}, x \neq y\}$.

Definition 11 ($\mathcal{L}$: the logic of well-formed formulae). *A formula φ is called* well-formed *if $Co \vdash \varphi$, for some $Co \subseteq Var \times Var$, where the entailment relation $\vdash$ is defined by the rules below:*

$$\frac{(\boldsymbol{x} \cup \boldsymbol{y}, \boldsymbol{x} \cup \boldsymbol{y}) \subseteq Co \quad Co \cup (z, \boldsymbol{x} \cup \boldsymbol{y}) \vdash \varphi}{Co \vdash (\boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\,z)\varphi} \qquad \frac{(\{z\}, fv(\varphi)) \in Co \quad Co \vdash \varphi}{Co \vdash \langle z\rangle\,\varphi}$$

$$\frac{Co \vdash \varphi}{Co \vdash \neg\varphi} \qquad \frac{Co \vdash \varphi_1 \quad Co \vdash \varphi_2}{Co \vdash \varphi_1 \wedge \varphi_2} \qquad \frac{}{Co \vdash \top}$$

We denote by $\mathcal{L}$ the logic consisting of the well-formed formulae.

According to the first rule, the formula $(\boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\,z)\varphi$ is well-formed if the constraints in Co ensure that $\boldsymbol{x} \cup \boldsymbol{y}$ is pairwise consistent and φ can be proved well-formed using also the fact that the chosen z will be consistent with $\boldsymbol{x}$ and $\boldsymbol{y}$. The second rule, instead, says that $\langle z\rangle\,\varphi$ is well-formed when the constraints in Co ensure that z is consistent with the free-variables used in φ and φ itself is well-formed in Co.

As an example, the formula $(\mathsf{a}\,x)(\mathsf{b}\ y)\langle x\rangle\,\langle y\rangle$ is not well-formed (since it "executes" x and y which, in principle, can be bound to events in conflict) as opposed to $(\mathsf{a}\,x)(x < \mathsf{b}\ y)\langle x\rangle\,\langle y\rangle$, where the two executed events are certainly consistent (they are causally dependent). Notice that the formula $(\mathsf{a}\,x)(x < \mathsf{b}\ y)\langle y\rangle\,\langle x\rangle$ is also well formed, even if it is always false since it tries to execute an event before executing one of its causes. Finally, according to the rules $(\mathsf{a}\, x)\langle x\rangle\ (\mathsf{b}\ y)\langle y\rangle$ is also deemed a well-formed formula even though there is no constraint on y. This can be understood recalling that the semantics of the binding operator requires y to be bound to an event that is consistent with the current state, hence consistent with the event bound to x.

When dealing with the semantics of formulae in $\mathcal{L}$ we will always consider environments η that reflect the corresponding consistency constraints.

Definition 12 (legal environment). *Let $\mathcal{E}$ be a* PES. *Given a configuration $C \in \mathcal{C}(\mathcal{E})$ and a formula φ in $\mathcal{L}$, a* legal environment *for C and φ is an environment $\eta : Var \to E$ such that $fv(\varphi) \subseteq dom(\eta)$, $\eta(fv(\varphi))$ is consistent with C and there exists Co such that $Co \vdash \varphi$ and $\forall (x, y) \in Co$, $\eta(x) \frown \eta(y)$.*

The semantics of the logic $\mathcal{L}$ is then formally defined as in Definition 10, where η is assumed to be a legal environment for C and φ. It is easy to see that this assumption properly fits with Definition 10, i.e., whenever we recur on a subformula, we are surely checking satisfiability in an environment legal for the configuration and the formula.

Definition 13 (logical equivalence). *Let $\mathcal{L}'$ be a fragment of $\mathcal{L}$. We say that two* PES *$\mathcal{E}_1, \mathcal{E}_2$ are* logically equivalent *in $\mathcal{L}'$, written $\mathcal{E}_1 \equiv_{\mathcal{L}'} \mathcal{E}_2$ when they satisfy the same closed formulae.*

3.1 Examples and Notation

In this subsection we provide some more examples illustrating the expressiveness of the logic. We start by introducing some handy notation, which will improve the readability of the formulae.

Immediate execution. We will write

$$\llparenthesis \boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\, z \rrparenthesis\, \varphi$$

for the formula $(\boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\, z)\langle z\rangle\, \varphi$ that chooses a consistent event not belonging to the current configuration, and immediately executes it.

Steps. We introduce a notation also to predicate the existence, resp., the immediate execution, of concurrent events, specifying also their dependencies. We will write

$$((\boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\, z) \otimes (\boldsymbol{x}', \overline{\boldsymbol{y}'} < \mathsf{b}\, z'))\, \varphi \qquad \text{and} \qquad (\llparenthesis \boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\, z \rrparenthesis \otimes \llparenthesis \boldsymbol{x}', \overline{\boldsymbol{y}'} < \mathsf{b}\, z' \rrparenthesis)\varphi$$

to declare the existence, resp., the immediate execution, of two concurrent events, labelled a and b, which are bound to z and z', and then φ holds. These notations correspond, respectively, to the formulae $(\boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\, z)(\boldsymbol{x}', \overline{\boldsymbol{y}', z} < \mathsf{b}\, z')\varphi$ and $(\,(\boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\, z) \otimes (\boldsymbol{x}', \overline{\boldsymbol{y}'} < \mathsf{b}\, z')\,)\langle z\rangle\, \langle z'\rangle\, \varphi$. In particular, the ability to perform a step consisting of two concurrent events labelled by a and b is simply expressed by the formula $\llparenthesis \mathsf{a}\, x \rrparenthesis \otimes \llparenthesis \mathsf{b}\, y \rrparenthesis$. Clearly, this notation can be generalised to the quantification and the immediate execution of any number of concurrent events.

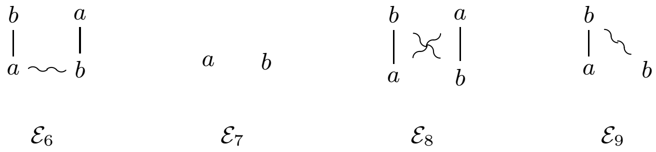
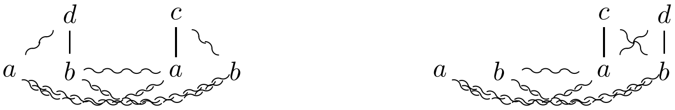


Fig. 2.

Example 1 (Interleaving vs. True-concurrency). As a first example, consider the PESs $\mathcal{E}_6$ and $\mathcal{E}_7$ in Fig. 2. They are equated by interleaving equivalences and kept distinct by any true-concurrent equivalence. The formula $\varphi_1 = \llparenthesis \mathsf{a}\, x \rrparenthesis \llparenthesis \overline{x} < \mathsf{b}\, y \rrparenthesis = (\llparenthesis \mathsf{a}\, x \rrparenthesis \otimes \llparenthesis \mathsf{b}\, y \rrparenthesis)$ is true only on $\mathcal{E}_7$, while $\varphi_2 = \llparenthesis \mathsf{a}\, x \rrparenthesis \llparenthesis x < \mathsf{b}\, y \rrparenthesis$ is true only on $\mathcal{E}_6$.

Example 2 (Dependent vs Independent Events). The need of considering both causal dependency and independency in the binding operator of our logic is exemplified below. Consider the PESs $\mathcal{E}_6$ and $\mathcal{E}_8$ in Fig. 2. They are distinguished by any true-concurrent equivalence, but since they have the same causal structure, in order to pinpoint how they differ, the logic must be able to express the presence of two concurrent events: indeed $\mathcal{E}_8 \models \llparenthesis \mathsf{a}\, x \rrparenthesis \otimes \llparenthesis \mathsf{b}\, y \rrparenthesis$, while $\mathcal{E}_6 \not\models \llparenthesis \mathsf{a}\, x \rrparenthesis \otimes \llparenthesis \mathsf{b}\, y \rrparenthesis$. On the other hand, expressing causal dependencies between events is essential to distinguish, for instance, the PESs $\mathcal{E}_7$ and $\mathcal{E}_9$. These are equated by step bisimulation and distinguished by any equivalence which observes causality, e.g., pomset bisimulation.

Example 3 (Conflicting Futures). Consider the following two PESs, which can be proved to be hp-bisimilar but not hhp-bisimilar:



Intuitively, they differ since the causes of the c-labelled and d-labelled events are in conflict in the first PES and independent in the second one. This is captured by the formula $\varphi = ((\mathsf{a}\, x) \otimes (\mathsf{b}\, y))((x < \mathsf{c}\, z_1) \wedge (y < \mathsf{d}\, z_2))$, which is satisfied only by the right-most PES. Notice that the formula φ exploits the ability of the logic $\mathcal{L}$ of quantifying over events in conflict with previously bound events: formula φ is satisfied in the rightmost PES by binding x and y to the rightmost a-labelled and b-labelled events; then both z_1 and z_2 are bound to events which are in conflict with either x or y. For this, the possibility of quantifying over an event without executing it is essential: the formula $\varphi' = (\llparenthesis \mathsf{a}\, x \rrparenthesis \otimes \llparenthesis \mathsf{b}\, y \rrparenthesis)((x < \mathsf{c}\, z_1) \wedge (y < \mathsf{d}\, z_2))$ would be false in both PESs since the execution of the first two events leads to a configuration that is no further extensible.

As a final example, consider the two CCS processes $P = a|(b{+}c) + a|b + b|(a{+}c)$ and $Q = a|(b+c) + b|(a+c)$. They contain no causal dependencies, but they exhibit a different interplay between concurrency and branching. Accordingly, the corresponding PESs can be proved to be hp-bisimilar but not hhp-bisimilar. Intuitively, this difference comes from the fact that only the process P includes two concurrent events a and b such that, independently from their execution order, no c-labelled event will be enabled. Such a difference can be expressed in $\mathcal{L}$ by the formula $((\mathsf{a}\, x) \otimes (\mathsf{b}\, y))(\neg(\overline{x} < \mathsf{c}\, z) \wedge \neg(\overline{y} < \mathsf{c}\, z'))$, which is satisfied only by the PES corresponding to P.

4 A Logical Characterisation of Concurrent Equivalences

In this section we study the logical equivalences induced by fragments of $\mathcal{L}$. We have already argued that no formula in $\mathcal{L}$ distinguishes the PESs a and $a\#a$, hence the logical equivalence induced by $\mathcal{L}$ is surely coarser than isomorphism. We next show that the logical equivalence induced by $\mathcal{L}$ is hhp-bisimulation. Moreover, we identify suitable fragments of $\mathcal{L}$ corresponding to coarser equivalences.

Theorem 1 (hhp-bisimilarity). *Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be* PES*s. Then $\mathcal{E}_1 \sim_{hhp} \mathcal{E}_2$ iff $\mathcal{E}_1 \equiv_{\mathcal{L}} \mathcal{E}_2$.*

4.1 From Hennessy-Milner Logic to HP-Logic

Hhp-bisimilarity is the finest equivalence in the spectrum of true concurrent equivalences in [2]. Coarser equivalences such as step, pomset and hp-bisimilarity, can be captured by suitable fragments of $\mathcal{L}$ summarised in Fig. 3, which can be viewed as the logical counterpart of the true concurrent spectrum.

Interestingly, in each of these fragments after predicating the existence of an event we must execute it. As a consequence, differently from what happens in the full logic, in the fragments it is impossible to refer to events in conflict with already observed events. Intuitively, behavioural equivalences up to hp-bisimilarity observe events only by executing them. Hence they cannot fully capture the interplay between concurrency and branching, which is indeed distinctive of hhp-equivalence.

HM Logic	$\mathcal{L}_{HM}$	$\varphi ::= \llparenthesis \mathsf{a}\, x \rrparenthesis\, \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \top$
Step Logic	$\mathcal{L}_s$	$\varphi ::= (\llparenthesis \mathsf{a}_1\, x_1 \rrparenthesis \otimes \cdots \otimes \llparenthesis \mathsf{a}_n\, x_n \rrparenthesis)\, \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \top$
Pomset Logic	$\mathcal{L}_p$	$\varphi ::= \llparenthesis \boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\, z \rrparenthesis\, \varphi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \top$ where $\neg$, $\wedge$ are used only on closed formulae.
HP Logic	$\mathcal{L}_{hp}$	$\varphi ::= \llparenthesis \boldsymbol{x}, \overline{\boldsymbol{y}} < \mathsf{a}\, z \rrparenthesis\, \varphi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \top$

Fig. 3. Fragments corresponding to behavioral equivalences

Hennessy-Milner Logic. A first simple observation is that Hennessy-Milner logic can be recovered as the fragment of $\mathcal{L}$ where only the derived modality $\llparenthesis \mathsf{a}\, x \rrparenthesis \varphi$ (with no references to causally dependent/concurrent events) is allowed. In words, whenever we state the existence of an enabled event we are forced to execute it. Moreover, since no dependencies can be expressed, the bound variable x is irrelevant. The induced logical equivalence is thus bisimilarity [3] (recall that we consider only image finite PES's).

Step logic. A fragment $\mathcal{L}_s$ corresponding to step bisimilarity naturally arises as a generalisation of HM logic, where we can refer to sets of concurrently enabled events. More precisely, as shown in Fig. 3, $\mathcal{L}_s$ is the fragment of $\mathcal{L}$ where only the derived modality $\llparenthesis \mathsf{a}_1\, x_1 \rrparenthesis \otimes \cdots \otimes \llparenthesis \mathsf{a}_\mathsf{n}\, x_n \rrparenthesis$ is used, allowing to predicate on the possibility of performing a parallel step, but without any reference to causal dependencies. Note that all formulae in $\mathcal{L}_s$ are closed, and thus environments are irrelevant in their semantics (as well as the names of the bound variables).

As an example, consider the two PESs $\mathcal{E}_6$ and $\mathcal{E}_7$ in Fig. 2. They are bisimilar but not step bisimilar since only $\mathcal{E}_7$ can execute the step consisting of a and b; accordingly, the formula $\llparenthesis \mathsf{a} \rrparenthesis \otimes \llparenthesis \mathsf{b} \rrparenthesis$ in $\mathcal{L}_s$ is true only on $\mathcal{E}_7$.

Theorem 2 (step bisimilarity). *Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be PESs. Then $\mathcal{E}_1 \sim_s \mathcal{E}_2$ iff $\mathcal{E}_1 \equiv_{\mathcal{L}_s} \mathcal{E}_2$.*

Pomset logic. The logic $\mathcal{L}_p$ for pomset bisimilarity consists of a fragment of $\mathcal{L}$ which essentially predicates about the possibility of executing pomset transitions. Even in $\mathcal{L}_p$ the events must be immediately executed when quantified, but it is now possible to refer to dependencies between events. However, propositional connectives (negation and conjunction) can be used only on closed formulae.

Intuitively, in this logic closed formulae characterize the execution of a pomset. Then, the requirement that the propositional operators are used only on closed (sub)formulae prevents pomset transitions from being causally linked to the events in the past. These ideas are formalised by the results below, starting with a lemma showing that the execution of a pomset can be characterized as a formula in $\mathcal{L}_p$.

Definition 14 (pomsets as formulae in $\mathcal{L}_p$). *Let $p_x = (\{x_1, \ldots, x_n\}, \leq_{p_x}, \lambda_{p_x})$ be a labelled poset, whose elements $\{x_1, \ldots, x_n\}$ are variables ordered by $\leq_{p_x}$. Given a formula $\varphi \in \mathcal{L}_p$, we denote by $\llparenthesis\, p_x \rrparenthesis \varphi$ the formula inductively defined as follows. If p_x is empty then $\llparenthesis\, p_x \rrparenthesis \varphi = \varphi$. If $p_x = p'_x \cup \{x\}$, where x is maximal with respect to $\leq_{p_x}$, then if $\boldsymbol{y} = \{x' \in p'_x \mid x' \leq_{p_x} x\}$, $\boldsymbol{z} = p'_x \setminus \boldsymbol{y}$, and $\lambda_{p_x}(x) = \mathsf{a}$, then $\llparenthesis\, p_x \rrparenthesis \varphi = \llparenthesis\, p'_x \rrparenthesis \llparenthesis \boldsymbol{y}, \overline{\boldsymbol{z}} < \mathsf{a}\, x \rrparenthesis \varphi$.*

Lemma 1 (pomsets in $\mathcal{L}_p$). *Let $\mathcal{E}$ be a PES and let $C \in \mathcal{C}(\mathcal{E})$ be a configuration. Given a labelled poset $p_x = (\{x_1, \ldots, x_n\}, \leq_{p_x}, \lambda_{p_x})$, then*

$$\mathcal{E}, C \models_\eta \llparenthesis\, p_x \rrparenthesis\, \varphi \quad \textit{iff}$$

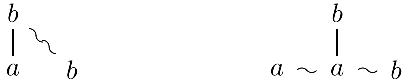
$$C \xrightarrow{X} C' \textit{ where } X = \{e_1, \ldots, e_n\} \textit{ is a pomset s.t. } X \sim p_x$$
$$\textit{and } \mathcal{E}, C' \models_{\eta'} \varphi, \textit{ with } \eta' = \eta[x_1 \mapsto e_1, \ldots, x_n \mapsto e_n]$$

As an example, consider the two PESs $\mathcal{E}_7$ and $\mathcal{E}_9$ in Fig. 2. They are step bisimilar but not pomset bisimilar since only the second one can execute the pomset $(\{\mathsf{a},\mathsf{b}\},\mathsf{a}<\mathsf{b})$. Accordingly, the formula $\varphi = \llparenthesis \mathsf{a}\, x \rrparenthesis \llparenthesis x < \mathsf{b}\, y \rrparenthesis$ in $\mathcal{L}_p$ is satisfied only by $\mathcal{E}_9$.

Theorem 3 (pomset bisimilarity). *Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be* PES*s. Then $\mathcal{E}_1 \sim_p \mathcal{E}_2$ iff $\mathcal{E}_1 \equiv_{\mathcal{L}_p} \mathcal{E}_2$.*

History preserving logic. The fragment $\mathcal{L}_{hp}$ corresponding to hp-bisimilarity is essentially the same as for pomset logic, where we relax the condition about closedness of the formulae the propositional connectives are applied to. Intuitively, in this way a formula $\varphi \in \mathcal{L}_{hp}$, besides expressing the possibility of executing a pomset p_x, also predicates about its dependencies with previously executed events (bound to the free variables of φ).

The two PESs below can be proved to be pomset equivalent but not hp-equivalent:



Intuitively, they allow the same pomset transitions, but they have a different "causal branching". Indeed, only in the left-most PESs after the execution of an a-labelled event we can choose between an independent and a dependent b-labelled event. Formally, the formula $\llparenthesis \mathsf{a}\, x \rrparenthesis (\llparenthesis \overline{x} < \mathsf{b}\, y \rrparenthesis \wedge (\llparenthesis x < \mathsf{b}\, z \rrparenthesis)$ in $\mathcal{L}_{hp}$ is true only on the left-most PES.

Theorem 4 (hp-bisimilarity). *Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be* PES*s. Then $\mathcal{E}_1 \sim_{hp} \mathcal{E}_2$ iff $\mathcal{E}_1 \equiv_{\mathcal{L}_{hp}} \mathcal{E}_2$.*

5 Conclusions: Related and Future Work

We have introduced a logic for true concurrency, which allows to predicate on events in computation and their mutual dependencies (causality and concurrency). The logic subsumes standard HM logic and provides a characterisation of the most widely known true concurrent behavioural equivalences: hhp-bisimilarity is the logical equivalence induced by the full logic, and suitable fragments are identified which induce hp-bisimilarity, pomset and step bisimilarity.

As we mentioned in the introduction, there is a vast literature relating logical and operational views of true concurrency, however, to the best of our knowledge, a uniform logical counterpart of the true concurrent spectrum is still missing. An exhaustive account of the related literature is impossible; we just recall here the approaches that most closely relate to our work.

In [5,15,16] the causal structure of concurrent systems is pushed into the logic. The paper [5] considers modalities which describe pomset transitions, thus providing an immediate characterization of pomset bisimilarity. Moreover, [5,15,16] show that by tracing the history of states and adding the possibility of reverting pomset transitions, one obtains an equivalence coarser than hp-bisimilarity and incomparable with pomset bisimilarity, called weak hp-bisimilarity. (We remark that, despite its name, weak hp-bisimilarity is not related to silent actions.) Our logic intends to be more general by also capturing the interplay between concurrency and branching, which is not observable at the level of hp-bisimilarity.

A recent work [8,9] introduces a fixpoint modal logic for true concurrent models, called Separation Fixpoint Logics (SFL). This includes modalities which specify the execution of an action causally dependent/independent on the last executed one. Moreover, a "separation operator" deals with concurrently enabled actions. The fragment of the logic without the separation operator is shown to capture hp-bisimilarity, while the full logic induces an equivalence which lies in between hp- and hhp-bisimilarity, still being decidable for finite state systems. The approach of [8,9] is inspired by the so-called Independence-Friendly Modal Logic (IFML) [13], which includes a modality that allows to specify that the current executed action is independent from a number of previously executed ones. In this sense IFML is similar in spirit to our logic. Although, most of the equivalences induced by fragments of IFML are not standard in the true concurrent spectrum, a deeper comparison with this approach represents an interesting open issue.

Several classical papers have considered temporal logics with modalities corresponding to the "retraction" or "backward" execution of computations. In particular [10,11,12] study a so-called path logic with a future perfect (also called past tense) modality: $@\mathsf{a}\,\varphi$ is true when φ holds in a state which can reach the current one with an a-transition. When interpreted over transition systems with independence, in absence of autoconcurrency, the logic characterises hhp-bisimilarity. In [10] it is shown that, taking autoconcurrency into account, the result can be extended at the price of complicating the logic (roughly, the logic needs an operator to undo a specific action performed in the past).

Compared to these works, the main novelty of our approach resides in the fact that the logic $\mathcal{L}$ provides a characterisation of the different equivalences in a simple, unitary logical framework. In order to enforce this view, we intend to pursue a formal comparison with the logics for concurrency introduced in the literature. It is easy to see that the execution modalities of [8,9] can be encoded in $\mathcal{L}$ since they only refer to the last executed event, while the formulae in $\mathcal{L}$ can refer to any event executed in the past. On the other hand, the "separation operator" of [8,9], as well as the backward modalities mentioned above (past tense, future perfect, reverting pomset transitions) are not immediately encodable in $\mathcal{L}$. A deeper investigation would be of great help in shading further light on the truly concurrent spectrum. Moreover $\mathcal{L}$ suggests an alternative, forward-only, operational definition of hhp-bisimulation, which could be inspiring also for other reverse bisimulations [17].

As a byproduct of such an investigation, we foresee the identification of interesting extensions of the concurrent spectrum, both at the logical and at the operational side. For instance, a preliminary investigation suggests that the fragment of $\mathcal{L}$ where only events consistent with the current environment can be quantified induces an equivalence which admits a natural operational definition and lies in between hp- and hhp-bisimilarity, still being different from the equivalences in [8,9]. Moreover, the logic in its present form only allows to describe properties of finite, bounded computations. A more powerful specification logic, well-suited for describing properties of unbounded, possibly infinite computations can be obtained by enriching $\mathcal{L}$ with some form of recursion. In particular, from some first experiments, the idea of "embedding" our logic into a first order modal mu-calculus in the style of [18,19] seems promising. For this purpose, also the fixpoint extension of the Independence-Friendly Modal Logic in [20] could be inspiring. The resulting logic would allow to express non-trivial causal properties, like "any a action can be always followed by a causally related b action in at most three steps", or "an a action can be always executed in parallel with a b action".

Connected to this, model-checking and decidability issues are challenging directions of future investigation (see [21] for a survey focussed on partial order temporal logics). It is known that hhp-bisimilarity is undecidable, even for finite state systems [22], while hp-bisimilarity is decidable. Characterising decidable fragments of the logic could be helpful in drawing a clearer separation line between decidability and undecidability of concurrent equivalences. A promising direction is to impose a bound on the "causal depth" of the future which the logic can quantify on. In this way one gets a chain of equivalences, coarser than hhp-bisimilarity, which should be closely related with n-hhp bisimilarities introduced and shown to be decidable in [23]. As for verification, we aim at investigating the automata-theoretic counterpart of the logic. In previous papers, hp-bisimilarity has been characterised in automata-theoretic terms using HD-automata [6] or Petri nets [7]. It seems that HD-automata [6] could provide a suitable automata counterpart of the fragment $\mathcal{L}_{hp}$. Also the game-theoretical approach proposed in [8,9] for the fixpoint separation logic can be a source of inspiration.

References

1. van Glabbeek, R.: The linear time – branching time spectrum I; the semantics of concrete, sequential processes. In: Handbook of Process Algebra, pp. 3–99. Elsevier, Amsterdam (2001)
2. van Glabbeek, R., Goltz, U.: Refinement of actions and equivalence notions for concurrent systems. Acta Informatica 37(4/5), 229–327 (2001)
3. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. Journal of the ACM 32, 137–161 (1985)
4. Winskel, G., Nielsen, M.: Models for concurrency. In: Handbook of logic in Computer Science, vol. 4. Clarendon Press, Oxford (1995)
5. De Nicola, R., Ferrari, G.: Observational logics and concurrency models. In: Veni Madhavan, C.E., Nori, K.V. (eds.) FSTTCS 1990. LNCS, vol. 472, pp. 301–315. Springer, Heidelberg (1990)
6. Montanari, U., Pistore, M.: Minimal transition systems for history-preserving bisimulation. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 413–425. Springer, Heidelberg (1997)

7. Vogler, W.: Deciding history preserving bisimilarity. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) ICALP 1991. LNCS, vol. 510, pp. 495–505. Springer, Heidelberg (1991)
8. Gutierrez, J., Bradfield, J.C.: Model-checking games for fixpoint logics with partial order models. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 354–368. Springer, Heidelberg (2009)
9. Gutierrez, J.: Logics and bisimulation games for concurrency, causality and conflict. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 48–62. Springer, Heidelberg (2009)
10. Nielsen, M., Clausen, C.: Games and logics for a noninterleaving bisimulation. Nordic Journal of Computing 2(2), 221–249 (1995)
11. Bednarczyk, M.A.: Hereditary history preserving bisimulations or what is the power of the future perfect in program logics. Technical report, Polish Academy of Sciences (1991)
12. Hennessy, M., Stirling, C.: The power of the future perfect in program logics. Information and Control 67(1-3), 23–52 (1985)
13. Bradfield, J., Fröschle, S.: Independence-friendly modal logic and true concurrency. Nordic Journal of Computing 9(1), 102–117 (2002)
14. Winskel, G.: Event Structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 255, pp. 325–392. Springer, Heidelberg (1987)
15. Pinchinat, S., Laroussinie, F., Schnoebelen, P.: Logical characterization of truly concurrent bisimulation. Technical Report 114, LIFIA-IMAG, Grenoble (1994)
16. Cherief, F.: Back and forth bisimulations on prime event structures. In: Etiemble, D., Syre, J.-C. (eds.) PARLE 1992. LNCS, vol. 605, pp. 843–858. Springer, Heidelberg (1992)
17. Phillips, I., Ulidowski, I.: Reverse bisimulations on stable configuration structures. In: Proc. of SOS 2009. Electronic Proceedings in Theoretical Computer Science, vol. 18, pp. 62–76 (2010)
18. Dam, M.: Model checking mobile processes. Information and Computation 129(1), 35–51 (1996)
19. Dam, M., Fredlund, L.Å., Gurov, D.: Toward parametric verification of open distributed systems. In: de Roever, W.-P., Langmaack, H., Pnueli, A. (eds.) COMPOS 1997. LNCS, vol. 1536, pp. 150–185. Springer, Heidelberg (1998)
20. Bradfield, J., Kreutzer, K.: The complexity of independence-friendly fixpoint logic. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 355–368. Springer, Heidelberg (2005)
21. Penczek, W.: Branching time and partial order in temporal logics. In: Time and Logic: A Computational Approach, pp. 179–228. UCL Press, London (1995)
22. Jurdzinski, M., Nielsen, M., Srba, J.: Undecidability of domino games and hhp-bisimilarity. Information and Computation 184(2), 343–368 (2003)
23. Fröschle, S., Hildebrandt, T.: On plain and hereditary history-preserving bisimulation. In: Kutyłowski, M., Wierzbicki, T., Pacholski, L. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 354–365. Springer, Heidelberg (1999)

A Theory of Design-by-Contract for Distributed Multiparty Interactions*

Laura Bocchi[1], Kohei Honda[2], Emilio Tuosto[1], and Nobuko Yoshida[3]

[1] University of Leicester
[2] Queen Mary
University of London
[3] Imperial College London

Abstract. Design by Contract (DbC) promotes reliable software development through elaboration of type signatures for sequential programs with logical predicates. This paper presents an assertion method, based on the π-calculus with full recursion, which generalises the notion of DbC to multiparty distributed interactions to enable effective specification and verification of distributed multiparty protocols. Centring on *global assertions* and their *projections* onto endpoint assertions, our method allows clear specifications for typed sessions, constraining the *content* of the exchanged messages, the *choice* of sub-conversations to follow, and *invariants* on recursions. The paper presents key theoretical foundations of this framework, including a sound and relatively complete compositional proof system for verifying processes against assertions.

1 Introduction

This paper introduces an assertion method for specifying and verifying distributed multiparty interactions, drawing ideas from a framework known as Design-by-Contract (DbC), which is widely used in practice for sequential computation [13,18]. DbC [25] specifies a contract between a user and a sequential program by elaborating the type signature of the program with pre/post-conditions and invariants. Instead of saying "the method fooBar should be invoked with a string and an integer: then it will return (if ever) another string", DbC allows us to say "the method fooBar should be invoked with a string representing a date d between 2007 and 2008 and an integer n less than 1000 then it will (if ever) return a string representing the date n days after d". A type signature describes a basic shape of how a program can interact with other programs, stipulating its key *interface* to other components, which may be developed by other programmers. By associating signatures with logical predicates, DbC enables a highly effective framework for specifying, validating and managing systems' behaviour, usable throughout all phases of software development [21,23,28]. As a modelling and programming practice, DbC encourages engineers to make contracts among software modules precise [14,25], and build a system on the basis of these contracts.

The traditional DbC-based approaches are however limited to type signature of sequential procedures. A typical distributed application uses interaction scenarios that are

* This work is partially supported by EPSRC EP/F003757, EP/F002114, EP/G015635 and EP/G015481, Leverhulme Trust Award "Tracing Networks".

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 162–176, 2010.

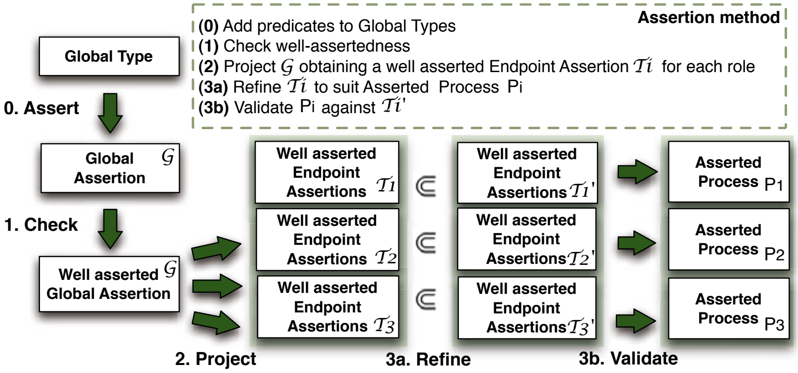

Assertion method
(0) Add predicates to Global Types
(1) Check well-assertedness
(2) Project $\mathcal{G}$ obtaining a well asserted Endpoint Assertion $\mathcal{T}i$ for each role
(3a) Refine $\mathcal{T}i$ to suit Asserted Process Pi
(3b) Validate Pi against $\mathcal{T}i'$
Global Type
0. Assert
Global Assertion $\mathcal{G}$
1. Check
Well asserted Global Assertion $\mathcal{G}$
Well asserted Endpoint Assertions $\mathcal{T}1$
Well asserted Endpoint Assertions $\mathcal{T}2$
Well asserted Endpoint Assertions $\mathcal{T}3$
$\Subset$
Well asserted Endpoint Assertions $\mathcal{T}1'$
Well asserted Endpoint Assertions $\mathcal{T}2'$
Well asserted Endpoint Assertions $\mathcal{T}3'$
Asserted Process P1
Asserted Process P2
Asserted Process P3
2. Project
3a. Refine
3b. Validate

Fig. 1. The assertion method

much more complex than, say, request-reply. To build a theory that extends the core idea of DbC to distributed applications, we consider a generalised notion of type signature for distributed interactions centring on abstraction units, called *sessions*. A session consists of a structured series of message exchanges among multiple participants. Each session follows a stipulated protocol, given as a type called *session type* [3,19,20], which prescribes a conversation scenario among its participants. Two or more sessions can interleave in a single endpoint. For example, a session for an electronic commerce will run interleaved with one for a financial transaction for payment. The communications in a distributed application are articulated as a collection of such sessions.

On this basis, we introduce a theory of assertions for distributed interactions centring on *global assertions*. A global assertion specifies a contract for participants in a multiparty session by elaborating a session type with logical predicates. A session type only specifies a skeletal protocol: it does not, for example, refer to constraints on message values except their types. Just as in the traditional DbC, the use of logical predicates allows us to specify more refined protocols, regarding, among others, *content* of messages, how *choice* of sub-conversations is made based on preceding interactions, and what *invariants* may be obeyed in recursive interactions. The key ideas are presented in Figure 1, which we illustrate below.

(0,1) A specification for a multiparty session is given as a *global assertion* $\mathcal{G}$, namely a protocol structure annotated with logical predicates. A minimal semantic criterion, *well-assertedness of* $\mathcal{G}$ (§ 3.1), characterises consistent specifications with respect to the temporal flow of events, to avoid unsatisfiable specifications.

(2) $\mathcal{G}$ is projected onto endpoints, yielding one *endpoint assertion* ($\mathcal{T}_i$) for each participant, specifying the behavioural responsibility of that endpoint (§ 4). The consistency of endpoint assertions are automatically guaranteed once the original global assertion is checked to be well-asserted.

(3) Asserted processes, modelled with the π-calculus[1] annotated with predicates (§ 5.1), are verified against endpoint assertions (§ 5.2) through a sound and relatively

[1] For the sake of a simpler presentation, the present paper does not treat name passing in full generality, except for private channel passing in session initiation. The theory however can incorpoate these elements, as explained in Section 7.

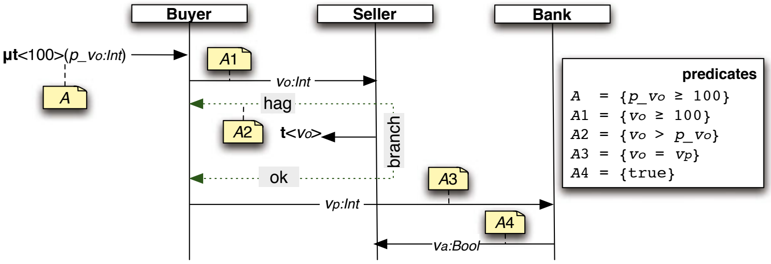


Fig. 2. Global assertion for the protocol

complete compositional proof system (§ 6). Completeness, proved through generation of principal formulae, yields a relative decision procedure for satisfiability.

Our contributions include an algorithmic validation of consistency of global assertions (Prop. 3.2 and 4.3); semantic foundations of global assertions through labelled transitions (Prop. 6.4 and 6.3); a compositional proof system for validating processes against assertions (Theorem 6.5), leading to *predicate error freedom* (Theorem 6.6) which ensures that the process will meet its obligations assuming that the remaining parties do so. Theorem 6.7 is completeness. § 7 concludes with further results and related work.

2 DbC for Distributed Multiparty Interactions

The theory we present centres on the notion of *global assertion*. A global assertion uses logical formulae to prescribe, for each interaction specified in the underlying session type, what the sending party must guarantee, and dually what the receiving party can rely on. Concretely:

1. Each message exchange in a session is associated with a predicate which constrains the values carried in the message (e.g., "the quantity on the invoice from seller to buyer equals the quantity on the order");
2. Each branch in a session is associated with a predicate which constrains the selection of that branch (e.g., "seller chooses the 'sell' option for a product if the ordered quantity does not exceed the stock");
3. Each recursion in a session is associated with an invariant representing an obligation to be maintained by all parties at each repetition of the recursion (e.g., "while negotiating, seller and buyer maintain the price per unit about a fixed threshold").

As an illustration, Figure 2 describes a simple multiparty session among the participants `Buyer`, `Seller`, and `Bank` exchanging messages whose content is represented by the *interaction variables* v_o, v_p (of type Int) and v_a (of type Bool). `Buyer` asynchronously sends an offer v_o, then `Seller` selects either to recursively start negotiating (hag) or to accept the offer (ok). In the latter case, `Buyer` instructs `Bank` to make a payment v_p. Finally, `Bank` sends `Seller` an acknowledgement v_a. The recursion parameter p_v_o is initially set to 100 and, upon recursive invocation, it takes the value that v_o had in the previous recursive invocation. This allows us to compare the current content of v_o with the one of the previous recursion instance (cf. $A2$ below). In Figure 2, the recursion invariant A states that p_v_o is always greater or equal than 100; `Buyer` guarantees $A1$ which, dually, `Seller` relies upon; by $A2$, `Buyer` has to increase the price during negotiations until an agreement is reached; the value of the (last) offer and the payment must be equal by $A3$, while $A4$ does not constrain v_a.

3 Global Assertions

We use the syntax of logical formulae, often called *predicates*, as follows.

$$A, B \ ::= \ e_1 = e_2 \ \mid \ e_1 > e_2 \ \mid \ \phi(e_1, \ldots, e_n) \ \mid \ A \wedge B \ \mid \ \neg A \ \mid \ \exists v(A)$$

where e_i ranges over expressions (which do not include channels) and ϕ over predefined atomic predicates with fixed arities and types [24, §2.8]. We denotes the set of *free variables* of A with $var(A)$, similarly for $var(e)$. We fix a model of the predicates, called *underlying logic*, for which we assume the validity of closed formulae to be decidable.

Global assertions (ranged over by $\mathcal{G}, \mathcal{G}', \ldots$) elaborate global session types in [20] with logical formulae. The syntax is given below:

$$\begin{array}{rll} \mathcal{G} \ ::= & \mathsf{p} \to \mathsf{p}' : k\,(\tilde{v} : \tilde{S})\{A\}.\mathcal{G} & \mid \ \mathbf{t}\langle \tilde{e} \rangle \\ \mid & \mathsf{p} \to \mathsf{p}' : k\,\{\{A_j\}l_j : \mathcal{G}_j\}_{j \in J} & \mid \ \mathcal{G}, \mathcal{G}' \\ \mid & \mu \mathbf{t}\langle \tilde{e} \rangle(\tilde{v} : \tilde{S})\{A\}.\mathcal{G} & \mid \ \mathsf{end} \end{array}$$

- p, p',.. are *participants*,
- k, k',.. are *channels*,
- u, v,.. are *interaction variables*,
- S, S',.. are *sorts*.

Interaction $\mathsf{p} \to \mathsf{p}' : k\,(\tilde{v} : \tilde{S})\{A\}.\mathcal{G}$ describes a communication between a sender p and a receiver p' via the k^{th} session channel (k is a natural number), followed by $\mathcal{G}$. The variables in the vector $\tilde{v}$ are called *interaction variables* and bind their occurrences in A and $\mathcal{G}$; interaction variables are sorted by *sorts* S (Bool, Int, ...) that denote types for first-order message values. The predicate A constrains the content of $\tilde{v}$: the sender p *guarantees* A and the receiver p' *relies* on A (like in the rely-guarantee paradigm [22]).

Branching $\mathsf{p} \to \mathsf{p}' : k\,\{\{A_j\}l_j : \mathcal{G}_j\}_{j \in J}$ allows the selector p to send to participant p', through k, a label l_i from $\{l_j\}_{j \in J}$ (J is a finite set of indexes) if p guarantees A_i (upon which p' can rely). Once l_i is selected, $\mathcal{G}_i$ is to be executed by all parties.

Recursive assertion $\mu \mathbf{t}\langle \tilde{e} \rangle(\tilde{v} : \tilde{S})\{A\}.\mathcal{G}$ (cf. [11], $\mathbf{t}$ is an *assertion variable*) specifies how a recursive session, which may be repeated arbitrarily many times, should be carried out through interactions among participants. The formal parameters $\tilde{v}$ are a vector of pairwise distinct variables sorted by a vector of sorts $\tilde{S}$ of the same length (each v_i in $\tilde{v}$ has sort S_i of $\tilde{S}$); $\tilde{v}$ binds their free occurrences in A. The *initialisation vector* $\tilde{e}$ denotes the initial values for the recursion, each e_i instantiating v_i in $\tilde{v}$. The *recursion invariant* A specifies the condition that needs be obeyed at each recursion instantiation; *recursion instantiation*, of the form $\mathbf{t}\langle \tilde{e} \rangle$, is to be guarded by prefixes, i.e. the underlying recursive types should be contractive. A recursive assertion can be unfolded to an infinite tree, as in the *equi-recursive* view on recursive types [30].

Composition $\mathcal{G}, \mathcal{G}'$ represents the parallel interactions specified by $\mathcal{G}$ and $\mathcal{G}'$, while end represents the termination. Sorts and trailing occurrences of end are often omitted.

We write $\mathsf{p} \in \mathcal{G}$ when p occurs in $\mathcal{G}$. For the sake of simplicity we avoid linearity-check [3] by assuming that each channel in $\mathcal{G}$ is used (maybe repeatedly) only between two parties: one party for input/branching and by the other for output/selection.

Example 3.1 (Global Assertions). The protocol described in § 2 is modelled by

$$\begin{array}{lll} \mathcal{G}_{neg} & = \mu\mathbf{t}\langle 100\rangle(p\_v_o : \mathsf{Int})\{A\}.\ \texttt{Buyer} \to \texttt{Seller}\colon k_1\,(v_o : \mathsf{Int})\{A1\}. \\ & \qquad\qquad\qquad\qquad\qquad\quad \texttt{Seller} \to \texttt{Buyer}\colon k_2\{\{A2\}\textbf{hag}\colon \mathbf{t}\langle v_o\rangle, \{\mathsf{true}\}\textbf{ok}\colon \mathcal{G}_{ok}\} \\ \mathcal{G}_{ok} & = \texttt{Buyer} \to \texttt{Bank}\colon k_3\,(v_p : \mathsf{Int})\{A3\}.\ \texttt{Bank} \to \texttt{Seller}\colon k_4\,(v_a : \mathsf{Bool})\{A4\}.\ \mathsf{end} \end{array}$$

where k_1, k_2, k_3, and k_4 are channels and the recursion parameter p_v_o (initially set to 100) denotes the offer of `Buyer` in the previous recursion instance.

3.1 Well Asserted Global Assertions

When setting up global assertions as a contract among multiple participants, we should prevent inconsistent specifications, such as those in which it is logically impossible for a participant to meet the specified obligations. Below we give two constraints on predicates of global assertions that guarantee consistency.

Let $I(\mathcal{G})$ be the set of variables occurring in $\mathcal{G}$; a participant $\mathtt{p}$ *knows* $v \in I(\mathcal{G})$ if v occurs in an interaction of $\mathcal{G}$ involving $\mathtt{p}$ (this relation can be computed effectively, see [31]). $I(\mathcal{G}) \upharpoonright \mathtt{p}$ denotes the set of variables of $\mathcal{G}$ that $\mathtt{p} \in \mathcal{G}$ knows.

History-sensitivity. A predicate guaranteed by a participant $\mathtt{p}$ can only contain those interaction variables that $\mathtt{p}$ knows.

Temporal-satisfiability. For each possible set of values satisfying A and, for each predicate A' appearing after A, it is possible to find values satisfying A'.

Consider the following examples:

$$\mathtt{p_A} \to \mathtt{p_B}\colon k_1(v : \mathsf{Int})\{\mathsf{true}\}.\ \mathtt{p_B} \to \mathtt{p_C}\colon k_2\,(v' : \mathsf{Int})\{\mathsf{true}\}.\ \mathtt{p_C} \to \mathtt{p_A}\colon k_3\,(z : \mathsf{Int})\{z > v\}.\ \mathsf{end}$$

$$\mathtt{p_A} \to \mathtt{p_B}\colon k_1(v : \mathsf{Int})\{v < 10\}\ \mathtt{p_B} \to \mathtt{p_A}\colon k_2\,(z : \mathsf{Int})\{v > z \wedge z > 6\}.\ \mathsf{end}.$$

The first global assertion violates history-sensitivity since $\mathtt{p_C}$ has to send z such that $z > v$ without knowing v. The second global assertion violates temporal-satisfiability because if $\mathtt{p_A}$ sends $v = 6$, which satisfies $v < 10$, then $\mathtt{p_B}$ will not be able to find a value that satisfies $6 > z \wedge z > 6$.

Assertions satisfying history-sensitivity and temporal-satisfiability are called *well-asserted assertions*. For the formal definitions, including inductive rules to check well-assertedness, see [31].

Proposition 3.2 (Well-assertedness). *Checking well-assertedness of a given global assertion is decidable if the underlying logic is decidable.*

4 Endpoint Assertions and Projection

Endpoint assertions, ranged over by $\mathcal{T}, \mathcal{T}', ..$, specify the behavioural contract of a session from the perspective of a single participant. The grammar is given as follows.

$$\begin{array}{llll} \mathcal{T} ::= & k!(\tilde{v}\colon \tilde{S})\{A\};\mathcal{T} & \mid \mu\mathbf{t}\langle\tilde{e}\rangle(\tilde{v}:\tilde{S})\{A\}.\mathcal{T} & \mid k\&\{\{A_i\}l_i\colon \mathcal{T}_i\}_{i\in I} \\ \mid & k?(\tilde{v}\colon \tilde{S})\{A\};\mathcal{T} & \mid \mathbf{t}\langle\tilde{e}\rangle & \mid k\oplus\{\{A_j\}l_j\colon \mathcal{T}_j\}_{j\in I} \mid \mathsf{end} \end{array}$$

In $k!(\tilde{v}\colon \tilde{S})\{A\};\mathcal{T}$, the sender guarantees that the values sent via k (denoted by $\tilde{S}$-sorted variables $\tilde{v}$) satisfy A, then behaves as $\mathcal{T}$; dually for the receiver $k?(\tilde{v}\colon \tilde{S})\{A\};\mathcal{T}$.

In $k \oplus \{\{A_j\}l_j\colon \mathcal{T}_j\}_{j\in I}$ the selector guarantees A_j when choosing l_j on k; dually $k\&\{\{A_j\}l_j\colon \mathcal{T}_j\}_{i\in I}$ states that A_j can be assumed when branching at k on a label l_j. Assertion $\mu\mathbf{t}\langle\tilde{e}\rangle(\tilde{v} : \tilde{S})\{A\}.\mathcal{T}$ constrains parameters $\tilde{v}$ of type $\tilde{S}$ which initially take values $\tilde{e}$; the invariant of the recursion is A.

The projection of predicate A on participant $\mathtt{p}$, written $A \upharpoonright \mathtt{p}$, existentially quantifies all the variables of A that $\mathtt{p}$ does not know, and is defined as $\exists V_{ext}(A)$ where $V_{ext} = var(A) \backslash \mathcal{I}(\mathcal{G}) \upharpoonright \mathtt{p}$. Also, $\tilde{e} \upharpoonright \mathtt{p}$ are the expressions in $\tilde{e}$ including only such that $var(e_i) \subseteq \mathcal{I}(\mathcal{G}) \upharpoonright \mathtt{p}$. The *projection* function in Definition 4.1 maps global assertions, predicates and participants to endpoint assertions.

Definition 4.1 (Projection). Given $\mathcal{G}$ and A, the *projection of $\mathcal{G}$ for a participant* $\mathtt{p}$ *wrt* A is denoted by $(\mathcal{G}) \downarrow_{\mathtt{p}}^{A}$ and, assuming $\mathtt{p}_1 \neq \mathtt{p}_2$, recursively defined as follows.

$$(1)\quad (\mathtt{p}_1 \to \mathtt{p}_2 \colon k\,(\tilde{v}\colon \tilde{S})\{A\}.\mathcal{G}') \downarrow_{\mathtt{p}}^{A_P} = \begin{cases} k!(\tilde{v}\colon \tilde{S})\{A\}.(\mathcal{G}') \downarrow_{\mathtt{p}}^{A\wedge A_P} & \text{if } \mathtt{p} = \mathtt{p}_1 \\ k?(\tilde{v}\colon \tilde{S})\{(A \wedge A_P) \upharpoonright \mathtt{p}\}.(\mathcal{G}') \downarrow_{\mathtt{p}}^{A\wedge A_P} & \text{if } \mathtt{p} = \mathtt{p}_2 \\ (\mathcal{G}') \downarrow_{\mathtt{p}}^{A\wedge A_P} & \text{otw} \end{cases}$$

$$(2)\quad (\mathtt{p}_1 \to \mathtt{p}_2 \colon k\,\{\{A_i\}l_i\colon\, \mathcal{G}_i\}_{i\in I}) \downarrow_{\mathtt{p}}^{A_P} = \begin{cases} k \oplus \{\{A_i\}l_i\colon\, (\mathcal{G}_i) \downarrow_{\mathtt{p}}^{A_i \wedge A_P}\}_{i\in I} & \text{if } \mathtt{p} = \mathtt{p}_1 \\ k\&\{\{(A_i \wedge A_P) \upharpoonright \mathtt{p}\}l_i\colon\, (\mathcal{G}_i) \downarrow_{\mathtt{p}}^{A_i \wedge A_P}\}_{i\in I} & \text{if } \mathtt{p} = \mathtt{p}_2 \\ (\mathcal{G}_1) \downarrow_{\mathtt{p}}^{A_P \wedge \bigvee_{j\in I} A_j} \;\; (= (\mathcal{G}_i) \downarrow_{\mathtt{p}}^{A_P \wedge \bigvee_{j\in I} A_j}) & \text{otw} \end{cases}$$

$$(3)\quad (\mathcal{G}_1, \mathcal{G}_2) \downarrow_{\mathtt{p}}^{A_P} = \begin{cases} (\mathcal{G}_i) \downarrow_{\mathtt{p}}^{A_P} & \text{if } \mathtt{p} \in \mathcal{G}_i \text{ and } \mathtt{p} \notin \mathcal{G}_j, i \neq j \in \{1,2\} \\ \mathsf{end} & \text{if } \mathtt{p} \notin \mathcal{G}_1 \text{ and } \mathtt{p} \notin \mathcal{G}_2 \end{cases}$$

$$(4)\quad (\mu\mathbf{t}\langle\tilde{e}\rangle(\tilde{v}\colon \tilde{S})\{A\}.\mathcal{G}) \downarrow_{\mathtt{p}}^{A_P} = \mu\mathbf{t}\langle\tilde{e} \upharpoonright \mathtt{p}\rangle(\tilde{v} \upharpoonright \mathtt{p}\colon S)\{A \upharpoonright \mathtt{p}\}.(\mathcal{G}) \downarrow_{\mathtt{p}}^{A_P}$$

$$(5)\quad (\mathbf{t}\langle\tilde{e}\rangle) \downarrow_{\mathtt{p}}^{A_P} = \mathbf{t}\langle\tilde{e} \upharpoonright \mathtt{p}\rangle \qquad\qquad (6)\ (\mathsf{end}) \downarrow_{\mathtt{p}}^{A_P} = \mathsf{end}$$

If no side condition applies, $(\mathcal{G}) \downarrow_{\mathtt{p}}^{A}$ is undefined. *The* projection of $\mathcal{G}$ on $\mathtt{p}$, denoted $\mathcal{G} \upharpoonright \mathtt{p}$, is given as $(\mathcal{G}) \downarrow_{\mathtt{p}}^{\mathsf{true}}$.

In (1), value passing interactions are projected. For a send, the projection of a predicate A consists of A itself. Notice that if $\mathcal{G}$ is well-asserted then $\mathtt{p}_1$ knows all variables in A (i.e., $A \upharpoonright \mathtt{p}_1 = A$). For a receive, it is not sufficient to verify the non-violation of the current predicate only. Consider the following well-asserted global assertion:

$$\mathtt{Seller} \to \mathtt{Buyer}\colon k_1\,(cost : \mathsf{Int})\{cost > 10\}.\mathtt{Buyer} \to \mathtt{Bank}\colon k_2\,(pay : \mathsf{Int})\{pay \geqslant cost\}.\mathsf{end}$$

The predicate $pay \geqslant cost$ is meaningless to Bank since Bank does not know *cost*; rather the projection on Bank should be $k_2?(pay : \mathsf{Int})\{\exists cost(cost > 10 \wedge pay \geqslant cost)\}$, which incorporates the constraint between Buyer and Seller. Thus (1) projects all the past predicates while hiding incorporating the constraints on interactions $\mathtt{p}_2$ does not participate through existential quantification. This makes (1) the strongest precondition i.e. it is satisfied iff $\mathtt{p}_2$ receives a legal message, avoiding the burden of defensive programming (e.g. the programmer of Bank can concentrate on the case $pay \leqslant 10$).

In (2), the "otw" case says the projection should be the same for all branches. In (3), each participant is in at most a single global assertion to ensure each local assertion is single threaded. In (4), the projection to p is the recursive assertion itself with its predicate projected on p by existential quantification, similarly in (5).

Example 4.2 (Projection). The projection of $\mathcal{G}_{neg}$ (Example 3.1) on `Seller` is

$$\begin{array}{lcl} \mathcal{T}_{sel} & = & \mu\mathbf{t}\langle 100\rangle(p\_v_o : \mathsf{Int})\{p\_v_o \geqslant 100\}; k_1?(v_o : \mathsf{Int})\{B\}; \mathcal{T}_2 \\ \mathcal{T}_2 & = & k_2 \oplus \{\{v_o > p\_v_o\}\mathbf{hag}: \mathbf{t}\langle v_o\rangle, \{\mathsf{true}\}\mathbf{ok}: \mathcal{T}_{ok}\} \\ \mathcal{T}_{ok} & = & \mathcal{G}_{ok} \upharpoonright \mathtt{Seller} = k_4?(v_a : \mathsf{Bool})\{B'\} \end{array}$$

where $B = p\_v_o \geqslant 100 \wedge v_o \geqslant 100$ and $B' = \exists p\_v_o.B \wedge v_o = v_p$.

Below well-assertedness can be defined on endpoint assertions as for global assertions, characterising the same two principles discussed in §3.1.

Proposition 4.3 (Projections). *Let $\mathcal{G}$ be a well-asserted global assertion. Then for each $\mathtt{p} \in \mathcal{G}$, if $\mathcal{G} \upharpoonright \mathtt{p}$ is defined then $\mathcal{G} \upharpoonright \mathtt{p}$ is also well-asserted.*

5 Compositional Validation of Processes

5.1 The π-Calculus with Assertions

We use the π-calculus with multiparty sessions [20, §2], augmented with predicates for checking (both outgoing and incoming) communications.

The grammar of *asserted processes* or simply *processes* $(P, Q, \ldots)$ is given below.

$$\begin{array}{llrllrrcl} P & ::= \overline{a}[\mathtt{2..n}](\tilde{s}).P & \text{request} & \mid & s \lhd \{A\}l; P & \text{select} & P_{rt} & ::= & P \mid (\nu\tilde{s})P_{rt} \\ & \mid a[\mathtt{p}](\tilde{s}).P & \text{accept} & \mid & s \rhd \{\{A_i\}l_i : P_i\}_{i\in I} & \text{branch} & & \mid & s : \tilde{h} \\ & \mid (\nu a)P & \text{hide} & \mid & P \mid Q & \text{parallel} & & \mid & \mathsf{errH} \mid \mathsf{errT} \\ & \mid s!\langle\tilde{e}\rangle(\tilde{v})\{A\}; P & \text{send} & \mid & \mu X\langle\tilde{e}\tilde{t}\rangle(\tilde{v}\tilde{s}).P & \text{rec def} & e & ::= & n \mid e \wedge e' \ldots \\ & \mid s?(\tilde{v})\{A\}; P & \text{receive} & \mid & X\langle\tilde{e}\tilde{s}\rangle & \text{rec call} & n & ::= & a \mid \mathsf{true} \mid \mathsf{false} \\ & \mid \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q & \text{conditional} & \mid & \mathbf{0} & \text{idle} & h & ::= & l \mid \tilde{n} \end{array}$$

On the left, we define *programs*. $\overline{a}[\mathtt{2..n}](\tilde{s}).P$ multicasts a session initiationrequest to each $a[\mathtt{p}](\tilde{s}).P$ (with $2 \leqslant \mathtt{p} \leqslant n$) by multiparty synchronisation through a *shared name* a. Send, receive, and selection, all through a *session channel* s, are associated with a predicate. Branch associates a predicate to each label. Others are standard.

Runtime processes P_{rt}, given in the third column in the grammar, extend programs with runtime constructs. Process $s : h_1..h_n$ represents messages in transit through a session channel s, assuming asynchronous in-order delivery as in TCP, with each h_i denoting either a branching label or a vector of sessions/values. The empty queue is written $s : \varnothing$. Processes errH and errT denote two kinds of run-time assertion violation: errH (for "error here") indicates a predicate violation by the process itself; and errT ("error there") a violation by the environment.

Example 5.1 (Seller's Process). We set `Buyer`, `Seller`, `Bank` to be participants `1`, `2`, `3` and define a process implementing the global assertion $\mathcal{G}_{neg}$ in Examples 3.1 and 4.2 as $P_{neg} = \overline{a}[\mathtt{2,3}](\tilde{s}).P_1 \mid a[\mathtt{2}](\tilde{s}).P_2 \mid a[\mathtt{3}](\tilde{s}).P_3$. Let us focus on the `Seller`

$$\begin{array}{lcl} P_2 & = & \mu X\langle 100, \tilde{s}\rangle(p\_v_o, \tilde{s}).s_1?(v_o)\{B\}; Q_2 \\ Q_2 & = & \mathsf{if}\ e\ \mathsf{then}\ (s_2 \lhd \mathbf{hag}; X\langle v_o, \tilde{s}\rangle)\ \mathsf{else}\ (s_2 \lhd \mathbf{ok}; P_{ok}) \quad \text{where} \quad P_{ok} = s_4?(v_a)\{B'\}; \mathbf{0} \end{array}$$

where B and B' are as in Example 4.2, $\tilde{s} = s_1, .., s_4$, and Q_2 uses a policy e to select a branch (e.g., $e = \{v_o > 200 \wedge v_o > p\_v_o\}$).

$$\overline{a}[2..\mathtt{n}](\tilde{s}).P_1 \mid a[2](\tilde{s}).P_2 \mid \ldots \mid a[\mathtt{n}](\tilde{s}).P_n \to (\nu\tilde{s})(P_1 \mid P_2 \mid \ldots \mid P_n \mid s_1:\varnothing \mid \ldots \mid s_n:\varnothing) \quad [\text{R-LINK}]$$

$$s!\langle\tilde{e}\rangle(\tilde{v})\{A\};P \mid s:\tilde{h} \to P[\tilde{\mathtt{n}}/\tilde{v}] \mid s:\tilde{h}\cdot\tilde{\mathtt{n}} \qquad (\tilde{e}\downarrow\tilde{\mathtt{n}} \wedge A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow \mathsf{true}) \quad [\text{R-SEND}]$$

$$s?(\tilde{v})\{A\};P \mid s:\tilde{\mathtt{n}}\cdot\tilde{h} \to P[\tilde{\mathtt{n}}/\tilde{v}] \mid s:\tilde{h} \qquad (A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow \mathsf{true}) \quad [\text{R-RECV}]$$

$$s \rhd \{\{A_i\}l_i : P_i\}_{i\in I} \mid s:l_j\cdot\tilde{h} \to P_j \mid s:\tilde{h} \qquad (j\in I \text{ and } A_j \downarrow \mathsf{true}) \quad [\text{R-BRANCH}]$$

$$s \lhd \{A\}l : P \mid s:\tilde{h} \to P \mid s:\tilde{h}\cdot l \qquad (A\downarrow \mathsf{true}) \quad [\text{R-SELECT}]$$

$$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \to P \ \ (e\downarrow\mathsf{true}) \quad \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \to Q \ \ (e\downarrow\mathsf{false}) \quad [\text{R-IF}]$$

$$s!\langle\tilde{e}\rangle(\tilde{v})\{A\};P \to \mathsf{errH} \quad (\tilde{e}\downarrow\tilde{\mathtt{n}} \wedge A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow \mathsf{false}) \quad [\text{R-SENDERR}]$$

$$s?(\tilde{v})\{A\};P \mid s:\tilde{\mathtt{n}}\cdot\tilde{h} \to \mathsf{errT} \mid s:\tilde{h} \quad (A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow \mathsf{false}) \quad [\text{R-RECVERR}]$$

$$s \rhd \{\{A_i\}l_i : P_i\}_{i\in I} \mid s:l_j\cdot\tilde{h} \to \mathsf{errT} \mid s:\tilde{h} \quad (j\in I \text{ and } A_j \downarrow \mathsf{false}) \quad [\text{R-BRANCHERR}]$$

$$s \lhd \{A\}l : P \to \mathsf{errH} \quad (A\downarrow \mathsf{false}) \quad [\text{R-SELECTERR}]$$

Fig. 3. Reduction: non-error cases (top) - error cases (bottom)

The reduction rules with predicate checking are given in Figure 3, which generate → by closing the induced relation under | and ν and taking terms modulo the standard structural equality[2] [20]. The satisfaction of the predicate is checked at each communication action: *send*, *receive*, *selection* and *branching*, where we write $A \downarrow \mathsf{true}$ (resp. $\tilde{e} \downarrow \tilde{\mathtt{n}}$) for a closed formula A (resp. expression $\tilde{e}$) when it evaluates to true (resp. $\tilde{\mathtt{n}}$). When initiating a session, [R-LINK] establishes a session through multiparty synchronisation, generating queues and hiding all session channels. The remaining rules are standard, modelling communications in a session via queues [3,20].

5.2 Validation Rules

For validation, we use judgements of the form $\mathcal{C};\Gamma \vdash P \rhd \Delta$, which reads: *"under* $\mathcal{C}$ *and* Γ*, process* P *is validated against* Δ*"*. Here, $\mathcal{C}$ is an *assertion environment*, which incrementally records the conjunction of predicates; hereafter, $\Gamma \vdash P \rhd \Delta$ abbreviates $\mathsf{true};\Gamma \vdash P \rhd \Delta$. Γ is a *global assertion assignment* that is a finite function mapping shared names to well-asserted global assertions and process variables to the specification of their parameters (we write $\Gamma \vdash a : \mathcal{G}$ when Γ assigns $\mathcal{G}$ to a and $\Gamma \vdash X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p}_1 \ldots \mathcal{T}_n @ \mathtt{p}_\mathtt{n}$ when Γ maps X to the vector of endpoint assertions $\mathcal{T}_1 @ \mathtt{p}_1 \ldots \mathcal{T}_n @ \mathtt{p}_\mathtt{n}$ using the variables $\tilde{v}$ sorted by $\tilde{S}$). Δ is an *endpoint assertion assignment* which maps the channels for each session, say $\tilde{s}$, to a well-asserted endpoint assertion located at a participant, say $\mathcal{T} @ \mathtt{p}$.

The validation rules are given in Figure 4. In each rule, we assume all occurring (global/endpoint) assertions to be well-asserted. The rules validate the process against assertions, simultaneously annotating processes with the interaction predicates from endpoint assertions. We illustrate the key rules.

Rule [SND] validates that participant $\mathtt{p}$ sends values $\tilde{e}$ on session channel k, *provided* that $\tilde{e}$ satisfy the predicate under the current assertion environment; and that the

[2] The structural equality includes $\mu X\langle\tilde{e}\rangle(\tilde{v}\tilde{s}_1\ldots\tilde{s}_n).P \equiv P[\mu X(\tilde{v}\tilde{s}_1\ldots\tilde{s}_n).P/X][\tilde{e}/\tilde{v}]$ where $X\langle\tilde{e}'\tilde{s}'\rangle[\mu X(\tilde{v}\tilde{s}_1\ldots\tilde{s}_n).P/X]$ is defined as $\mu X\langle\tilde{e}'\tilde{s}'\rangle(\tilde{v}\tilde{s}_1\ldots\tilde{s}_n).P$.

$$\frac{\mathcal{C} \supset A[\tilde{e}/\tilde{v}] \quad \mathcal{C};\Gamma \vdash P[\tilde{e}/\tilde{v}] \rhd \Delta, \tilde{s} : \mathcal{T}[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p} \quad \Gamma \vdash \tilde{e} : \tilde{S}}{\mathcal{C};\Gamma \vdash s_k!\langle\tilde{e}\rangle(\tilde{v} : \tilde{S})\{A\};P \rhd \Delta, \tilde{s} : k!(\tilde{v})\{A\};\mathcal{T} \mathbin{@} \mathtt{p}}\,[\textsc{Snd}]$$

$$\frac{\mathcal{C} \wedge A;\Gamma, \tilde{v} : \tilde{S} \vdash P \rhd \Delta,\ \tilde{s} : \mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C};\Gamma \vdash s_k?(\tilde{v} : \tilde{S})\{A\};P \rhd \Delta,\ \tilde{s} : k?(\tilde{v} : \tilde{S})\{A\};\mathcal{T} \mathbin{@} \mathtt{p}}\,[\textsc{Rcv}]$$

$$\frac{\mathcal{C} \supset A_j \quad \mathcal{C};\Gamma \vdash P \rhd \Delta, \tilde{s} : \mathcal{T}_j \mathbin{@} \mathtt{p} \quad j \in I}{\mathcal{C};\Gamma \vdash s_k \lhd \{A_j\}l_j : P \rhd \Delta,\ \tilde{s} : k \oplus \{\{A_i\}l_i : \mathcal{T}_i\}_{i \in I} \mathbin{@} \mathtt{p}}\,[\textsc{Sel}]$$

$$\frac{\mathcal{C} \wedge A_i;\Gamma \vdash P_i \rhd \Delta, \tilde{s} : \mathcal{T}_i \mathbin{@} \mathtt{p} \quad \forall i \in I}{\mathcal{C};\Gamma \vdash\ s_k \rhd \{\{A_i\}l_i : P_i\}_{i \in I} \rhd \Delta, \tilde{s} : k\&\{\{A_i\}l_i : \mathcal{T}_i\}_{i \in I} \mathbin{@} \mathtt{p}}\,[\textsc{Bra}]$$

$$\frac{\mathcal{C};\Gamma \vdash P \rhd \Delta, \tilde{s} : (\Gamma(a) \upharpoonright \mathtt{p}) \mathbin{@} \mathtt{p} \quad \mathtt{p} \gneq 1}{\mathcal{C};\Gamma \vdash a[\mathtt{p}]\,(\tilde{s}).P \rhd \Delta}\,[\textsc{Macc}] \qquad \frac{\mathcal{C};\Gamma \vdash P \rhd \Delta, \tilde{s} : (\Gamma(a) \upharpoonright 1) \mathbin{@} \mathtt{1}}{\mathcal{C};\Gamma \vdash \overline{a}[2..\mathtt{n}]\,(\tilde{s}).P \rhd \Delta}\,[\textsc{Mcast}]$$

$$\frac{\mathcal{C} \wedge e;\Gamma \vdash P \rhd \Delta \quad \mathcal{C} \wedge \neg e;\Gamma \vdash Q \rhd \Delta}{\mathcal{C};\Gamma \vdash \textsf{if } e \textsf{ then } P \textsf{ else } Q \rhd \Delta}\,[\textsc{If}] \quad \frac{\mathcal{C};\Gamma \vdash P \rhd \Delta \quad \mathcal{C};\Gamma \vdash Q \rhd \Delta'}{\mathcal{C};\Gamma \vdash P \mid Q \rhd \Delta, \Delta'}\,[\textsc{Conc}] \quad \frac{\Delta \textsf{ end only}}{\mathcal{C};\Gamma \vdash \mathbf{0} \rhd \Delta}\,[\textsc{Idle}]$$

$$\frac{\mathcal{C};\Gamma, a : \mathcal{G} \vdash P \rhd \Delta \quad a \notin \mathrm{fn}(\mathcal{C},\Gamma,\Delta)}{\mathcal{C};\Gamma \vdash (\nu a : \mathcal{G})P \rhd \Delta}\,[\textsc{Hide}] \qquad \frac{\mathcal{C}';\Gamma \vdash P \rhd \Delta' \quad \mathcal{C} \supset \mathcal{C}' \quad \Delta' \Supset \Delta}{\mathcal{C};\Gamma \vdash P \rhd \Delta}\,[\textsc{Conseq}]$$

$$\frac{\mathcal{T}_1[\tilde{e}/\tilde{v}], \ldots, \mathcal{T}_n[\tilde{e}/\tilde{v}] \ \text{ well-asserted and well-typed under } \Gamma, \tilde{v} : \tilde{S}}{\mathcal{C}\ ;\ \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 \mathbin{@} \mathtt{p}_1..\mathcal{T}_n \mathbin{@} \mathtt{p}_\mathtt{n} \vdash X\langle\tilde{e}\tilde{s}_1..\tilde{s}_n\rangle \rhd \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_n}\,[\textsc{Var}]$$

$$\frac{\mathcal{C}\ ;\ \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 \mathbin{@} \mathtt{p}_1..\mathcal{T}_n \mathbin{@} \mathtt{p}_n \vdash P \rhd \tilde{s}_1 : \mathcal{T}_1 \mathbin{@} \mathtt{p}_1..\tilde{s}_n : \mathcal{T}_n \mathbin{@} \mathtt{p}_n}{\mathcal{C}\ ;\ \Gamma \vdash \mu X\langle\tilde{e}\tilde{s}_1..\tilde{s}_n\rangle(\tilde{v}\tilde{s}_1..\tilde{s}_n).P \rhd\ \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_1..\tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_n}\,[\textsc{Rec}]$$

Fig. 4. Validation rules for program phrases

continuation is valid, once $\tilde{v}$ gets replaced by $\tilde{e}$. Dually, rule [Rcv] validates a value input against the continuation of the endpoint assertion under the extended assertion environment $\mathcal{C} \wedge A$ (i.e., the process can rely on A for the received values after the input). Rules [Sel] and [bra] are similar. Rules [Macc] and [Mcast] for session acceptance and request validate the continuation against the projection of the global assertion onto that participant ($\mathtt{n}$ is the number of participants in $\mathcal{G}$ and $\mathtt{p}$ is one of them).

Rule [If] validates a conditional against Δ if each branch is validated against the same Δ, under the extended environment $\mathcal{C} \wedge e$ or $\mathcal{C} \wedge \neg e$, as in the corresponding rule in Hoare logic. As in the underlying typing [20], rule [Conc] takes a disjoint union of two channel environments, and rule [Idle] takes Δ which only contains end as endpoint assertions. Rule [hide] is standard, assuming a is not specified in $\mathcal{C}$.

Rule [Conseq] uses the *refinement* relation $\Supset$ on endpoint assertions. If $\mathcal{T} \Supset \mathcal{T}'$, $\mathcal{T}$ specifies a *more refined behaviour* than $\mathcal{T}'$, in that $\mathcal{T}$ strengthens the predicates for send/selection, so it emits/selects less; and weakens those for receive/branching, so it can receive/accept more. Example 5.2 illustrates this intuition.

Example 5.2 (Refinement). Below, endpoint assertion $\mathcal{T}_s$ refines $\mathcal{T}_w$ (i.e., $\mathcal{T}_s \Supset \mathcal{T}_w$):

$$\begin{aligned} \mathcal{T}_s &= k_1!(v : \mathsf{Int})\{v > 10\};\ k_2?(z : \mathsf{Int})\{z > 0\};\ \ k_3\&\{\{\mathsf{true}\}\mathbf{l1} : \mathcal{T}_1, \{v > 100\}\mathbf{l2} : \mathcal{T}_2\} \\ \mathcal{T}_w &= k_1!(v : \mathsf{Int})\{v > 0\};\ \ k_2?(z : \mathsf{Int})\{z > 10\};\ k_3\&\{\{v > 100\}\mathbf{l1} : \mathcal{T}_1\} \end{aligned}$$

$\mathcal{T}_s$ has a stronger obligation on the sent value v, and a weaker reliance on the received value z; while $\mathcal{T}_s$ has a weaker guarantee at **l1** and offers one additional branch.

The formal definition is in [31], where we also show that the refinement relation is decidable if we restrict the use of recursive assertions so that only those in identical shapes are compared, which would suffice in many practical settings.

Rule [VAR] validates an instantiation of X with expressions against the result of performing the corresponding substitutions over endpoint assertions associated to X (in the environment). In [REC], a recursion is validated if the recursion body P is validated against the given endpoint assertions for its zero or more sessions, under the same endpoint assumptions assigned to the process variable X. The validity of this rule hinges on the partial correctness nature of the semantics of the judgement.

Example 5.3 (Validating Seller Process). We validate the `Seller` part of P_{neg} in Example 5.1 using $\mathcal{T}_{sel}$ from Example 3.1. We focus on one branch of Q_2 in P_{neg} and associate each $s_1,\ldots,s_4$ of P_{neg} to a channel $k_1,\ldots,k_4$ of $\mathcal{T}_{neg}$, respectively. Recall that $B = p\_v_o \geqslant 100 \land v_o \geqslant 100, A1 = v_o > p\_v_o$, and $A2 = \exists v_p.p\_v_o \geqslant 100 \land v_o \geqslant 100 \land v_o = v_p$. Below $Q_{ok} = s_4?(v_a)\{B'\};\mathbf{0}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{-}{(B \land \neg e \land B'),\Gamma \vdash \mathbf{0} \rhd t : \mathsf{end}\ @\ 2}\ [\textsc{Idle}]}{(B \land \neg e),\Gamma \vdash s_4?(v_a)\{B'\};\mathbf{0} \rhd \tilde{s} : k_4?(v_a : \mathsf{Int})\{B'\};\mathsf{end}\ @\ 2}\ [\textsc{Rcv}]}{(B \land \neg e) \supset A1 \quad (B \land \neg e),\Gamma \vdash Q_{ok} \rhd \tilde{s} : \mathcal{T}_{ok}\ @\ 2}\ \text{(substituting)}}{B \land \neg e,\Gamma \vdash s_2 \lhd \mathbf{ok};Q_{ok} \rhd \tilde{s} : k_2 \oplus \{\{\mathsf{true}\}\mathbf{ok} : \mathcal{T}_{ok}, \{A1\}\mathbf{hag} : \mathbf{t}\langle v_o\rangle\}\ @\ 2 \qquad \ldots}\ [\textsc{Sel}]\ [\textsc{If}]}{B,\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ (s_2 \lhd \mathbf{hag};X\langle v_o,\tilde{s}\rangle)\ \mathsf{else}\ (s_2 \lhd \mathbf{ok};s_4?(v_a)\{B'\};\mathbf{0}) \rhd \tilde{s} : \mathcal{T}_2\ @\ 2}}{\mathsf{true},\Gamma \vdash s_1?(v_o)\{B\};Q_2 \rhd \tilde{s} : k_1?(v_o : \mathsf{Int})\{B\};\mathcal{T}_2\ @\ 2}\ [\textsc{Rcv}]$$

The ... on the premise of [IF] indicates the missing validation of the first branch. The interested reader may refer to [31] for a complete validation example with recursion.

6 Error-Freedom and Completeness

6.1 Semantics of Assertions

The semantics of asserted processes is formalised as a labelled transition relation that uses the following labels

$$\alpha ::= \overline{a}[2..n](\tilde{s}) \mid a[i](\tilde{s}) \mid s!\tilde{n} \mid s?\tilde{n} \mid \mid s \lhd l \mid s \rhd l \mid \tau$$

for session requesting/accepting, value sending/receiving, selection, branching, and the silent action, respectively. We write $P \xrightarrow{\alpha} Q$ when P has a one-step transition α to Q. The transition rules are the standard synchronous ones[3] except that: (i) predicates are checked at each communication action and, if the predicate is violated, in the case of input/branching action the process moves to errT, in the case of an output/selection the process moves to errH with τ-action, (ii) they include the reduction semantics given in § 5.1 (i.e., $P \rightarrow Q$ induces $P \xrightarrow{\tau} Q$).

The semantics of endpoint assertions is defined as another labelled transition relation, of form $\langle\Gamma,\Delta\rangle \xrightarrow{\alpha} \langle\Gamma',\Delta'\rangle$, which reads: *the specification* $\langle\Gamma,\Delta\rangle$ allows *the action* α*, with* $\langle\Gamma',\Delta'\rangle$ *as the specification for its continuation.* In this transition relation, only legitimate (assertion-satisfying) actions are considered.

We define the semantic counterpart of $\Gamma \vdash P \rhd \Delta$ by using a simulation between the transitions of processes and those of assertions. The simulation (Definition 6.1),

[3] The synchronous transition suites our present purpose since it describes how a process places/retrieves messages at/from queues, when message content may as well be checked.

requires an input/branching action to be simulated only for "legal" values/labels, i.e. for actions in which predicates are not violated. Intuitively, we demand conformance to a proper behaviour only if the environment behaves properly. Below we use the *predicate erasure* to show that the validation can prevent bad behaviour even without runtime predicate checking, writing $\mathsf{erase}(P)$ for the result of erasing all predicates from P. Similarly $\mathsf{erase}(\Gamma)$ and $\mathsf{erase}(\Delta)$ erase predicates from the underlying session types, giving the *typing* environments. P is *closed* if it is without free variables.

Definition 6.1 (Conditional Simulation). Let $\mathcal{R}$ be a binary relation whose elements relate a closed process P without errH or errT and a pair of assignments $\langle\Gamma,\Delta\rangle$ such that $\mathsf{erase}(\Gamma) \vdash \mathsf{erase}(P) \rhd \mathsf{erase}(\Delta)$ in the typing rules in [20, §4]. Then $\mathcal{R}$ is a *conditional simulation* if, for each $(P, \langle\Gamma,\Delta\rangle) \in \mathcal{R}$:

1. for each input/branching/session input $P \xrightarrow{\alpha} P'$, $\langle\Gamma,\Delta\rangle$ has a respective move at $\mathsf{sbj}(\alpha)$ (the subject of α) and, if $\langle\Gamma,\Delta\rangle \xrightarrow{\alpha} \langle\Gamma',\Delta'\rangle$ then $(P', \langle\Gamma',\Delta'\rangle) \in \mathcal{R}$.
2. for each output/selection/τ/session output move $P \xrightarrow{\alpha} P'$, $\langle\Gamma,\Delta\rangle \xrightarrow{\alpha} \langle\Gamma',\Delta'\rangle$ such that $(P', \langle\Gamma',\Delta'\rangle) \in \mathcal{R}$.

If $\mathcal{R}$ is a conditional simulation we write $P \lesssim \langle\Gamma,\Delta\rangle$ for $(P, \langle\Gamma,\Delta\rangle) \in \mathcal{R}$.

The conditional simulation requires P to be well-typed against $\mathsf{erase}(\Gamma)$ and $\mathsf{erase}(\Delta)$. Without this condition, the inaction $\mathbf{0}$ would conditionally simulate any Δ. This stringent condition can be dropped, but our interest is to build an assertion semantics on the basis of the underlying type discipline.

Definition 6.2 (Satisfaction). Let P be a closed program and Δ an end-point assertion assignment. If $P \lesssim \langle\Gamma,\Delta\rangle$ then we say that P *satisfies* Δ *under* Γ, and write $\Gamma \models P \rhd \Delta$. The satisfaction is extended to open processes, denoted $\mathcal{C};\Gamma \models P \rhd \Delta$, by considering all closing substitutions respecting Γ and $\mathcal{C}$ over Δ and P.

The judgement $\Gamma \models P \rhd \Delta$ in Definition 6.2 states that (1) P will send valid messages or selection labels; and (2) P will continue to behave well (i.e., without going into error) w.r.t. the continuation specification after each valid action in (1) as well as after receiving each valid message/label (i.e. which satisfies an associated predicate). The satisfaction is about partial correctness since if P (is well-typed and) has no visible actions, the satisfaction trivially holds.

6.2 Soundness, Error Freedom and Completeness

To prove soundness of the validation rules, we first extend the validation rules to processes with queues, based on the corresponding typing rules in [3,20].

Proposition 6.3 (Subject Reduction). *Let* $\Gamma \vdash P \rhd \Delta$ *be a closed program and suppose we have* $\langle\Gamma,\Delta\rangle \overset{\alpha_1..\alpha_n}{\longrightarrow} \langle\Gamma',\Delta'\rangle$. *Then* $P \overset{\alpha_1..\alpha_n}{\longrightarrow} P'$ *implies* $\Gamma' \vdash P' \rhd \Delta'$.

The proof uses an analysis of the effects of τ-actions on endpoint assertions, observing the reduction at free session channels changes the shape of the session typing [3,20].

Let $\Delta \Supset \Delta'$ be a point-wise extension of $\Supset$ (defined when $\mathrm{dom}(\Delta) = \mathrm{dom}(\Delta')$); Proposition 6.4 says that a process satisfying a stronger specification also satisfies a weaker one. Using these results we obtain Theorem 6.5.

Proposition 6.4 (Refinement). *If* $\Gamma \models P \rhd \Delta$ *and* $\Delta \Supset \Delta'$ *then* $\Gamma \models P \rhd \Delta'$.

Theorem 6.5 (Soundness of Validation Rules). *Let P be a program. Then* $\mathcal{C};\Gamma \vdash P \rhd \Delta$ *implies* $\mathcal{C};\Gamma \models P \rhd \Delta$.

A direct consequence of Theorem 6.5 is the error freedom of validated processes. Below we say $\langle \Gamma, \Delta \rangle$ *allows* a sequence of actions $\alpha_1..\alpha_n$ $(n \geqslant 0)$ if for some $\langle \Gamma', \Delta' \rangle$ we have $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha_1..\alpha_n} \langle \Gamma', \Delta' \rangle$.

Theorem 6.6 (Predicate Error Freedom). *Suppose P is a closed program,* $\Gamma \vdash P \rhd \Delta$ *and* $P \xrightarrow{\alpha_1..\alpha_n} P'$ *such that* $\langle \Gamma, \Delta \rangle$ *allows* $\alpha_1..\alpha_n$. *Then P' contains neither* errH *nor* errT.

The proof system is complete relative to the decidability of the underlying logic for processes without hidden shared names. We avoid name restriction since it allows us to construct a process which is semantically equivalent to the inaction if and only if interactions starting from a hidden channel terminate. Since we can simulate arbitrary Turing machines by processes, this immediately violates completeness. In this case, non-termination produces a *dead code*, i.e. part of a process which does not give any visible action, which causes a failure in completeness.[4]

For each program without hiding, we can compositionally construct its "principal assertion assignment" from which we can always generate, up to $\Supset$, any sound assertion assignment for the process. Since the construction of principal specifications is compositional, it immediately gives an effective procedure to check $\models$ as far as $\Supset$ is decidable (which is relative to the underlying logic). We conclude:

Theorem 6.7 (Completeness of Validation Rules for Programs without Hiding). *For each closed program P without hiding, if* $\Gamma \models P \rhd \Delta$ *then* $\Gamma \vdash P \rhd \Delta$. *Further* $\Gamma \models P \rhd \Delta$ *is decidable relative to the decidability of* $\Supset$.

7 Extensions and Related Work

Extensions to shared and session channel passing. The theory we have introduced in the preceding sections directly extends to shared channel passing and session channel passing, or delegation, carrying over all formal properties. In both cases, we have only to add predicate annotations to channels in assertions as well as in asserted processes. The shape of the judgement and the proof rules do not change, similarly the semanics of the judgement uses a conditional simulation. We obtain the same soundness result as well as completeness of the proof rules for the class of processes whose newly created channels are immediately exported. Since the presentation of such extension would require a detailed presentation of the notion of refinement, for space constraints and simplicity of presentation we relegate it to [31].

Hennessy-Milner logic for the π-calculus. Hennessy-Milner Logic (HML) is an expressive modal logic with an exact semantic characterisation [17]. The presented theory addresses some of the key challenges in practical logical specifications for the π-calculus, unexplored in the context of HML. First, by starting from global assertions, we gain in significant concision of descriptions while enjoying generality within its scope (properties of individual protocols). Previous work [2,11] show how specifications in

[4] Not all dead codes cause failure in completeness. For example a dead branch in a branching/-conditional does not cause this issue since the validation rules can handle it.

HML, while encompassing essentially arbitrary behavioural properties of processes, tend to be lengthy from the practical viewpoint. In this context, the direct use of HML is tantamount to *reversing* the methodology depicted in Figure 1 of § 1: we start from endpoint specifications and later try to check their mutual consistency, which may not easily yield understandable global specifications.

As another practical aspect, since $\Supset$ is decidable for practically important classes assertions [31], the present theory also offers algorithmic validation methods for key engineering concerns [32] including consistency of specifications (cf. §3.1) and correctness of process behaviours with full recursion against non-trivial specifications (cf. Theorem 6.7), whose analogue may not be known for the general HML formulae on the π-calculus. The use of the underlying type structures plays a crucial role.

From the viewpoint of logical specifications for name passing, the present theory takes an *extensional* approach: we are concerned with what behaviours will unfold starting from given channels, than their (in)equality [11]. While our approach does reflect recommended practices in application-level distributed programming (where the direct use of network addresses is discouraged), it is an interesting topic to study how we can treat names as data as studied in [11].

Corresponding assertions and refinement/dependent types. The work [6] combines session-types with *correspondence assertions*. The type system can check that an assertion **end** L, where L is a list of values (not a logical formula), is matched by the corresponding **begin** effect.

The use of session types to describe behavioural properties of objects and components in CORBA is studied in [33]. In another vein, the refinement types for channels (e.g. [5]) specify value dependency with logical constraints. For example, one might write $?(x\colon \mathsf{int}, !\{y : \mathsf{int} \mid y > x\})$ using the notations from [15,34]. It specifies a dependency at a *single point* (channel), unable to describe a constraint for a series of interactions among multiple channels. Our theory, based on multiparty sessions, can verify processes against a contract globally agreed by multiple distributed peers.

Contract-based approaches to functions and communications and functions. Verification using theories of contracts for programming functional languages, with applications to the validation of financial contracts, is studied in [29,35]. Our theory uses the π-calculus with session types as the underlying formalism to describe contracts for distributed interactions. We observe that a contract-based approach for sequential computing is generally embeddable to the present framework (noting that function types are a special form of binary session types and that the pre/post conditions in sequential contracts are nothing but predicates for interactions resulting from the embedding); it is an interesting subject of study to integrate these and other sequential notions of contracts into the present framework, which would enable a uniform reasoning of sequential and concurrent processes.

In [8,12] use c-semirings to model constraints that specify a Service Level Agreement. It would be interesting to consider global assertions where the logical language is replaced with c-semirings. This would allow global assertions to express soft constraints but it could affect the effectiveness of our approach. However c-semirings do not feature negation and the decidability of logics based on c-semrings has not been deeply investigated.

The global consistency checking is used in advanced security formalisms. In [16] a rely-guarantee technique is applied to a trust-management logic. The main technical difference is that users have to directly annotate each participant with assertions because of the the absence of global assertions. In [4] cryptography is used to ensure integrity of sessions but logical contracts are not considered.

Theories of contracts for web services based on advanced behavioural types are proposed, including those using CCS [7], π-calculus [10], and conversation calculus [9]. Some of the authors in this line of study focus on *compliance* of client and services, often defining compliance in terms of deadlock-freedom, e.g., in [1] a type system guaranteeing a progress property of clients is defined.

Our approach differs from the preceding works in its use of global assertions for elaborating the underlying type structure, combined with the associated compositional proof system. This permits us to express and enforce fine-grained contracts of choreographic scenarios. Global/endpoint assertions can express constraints over message values (including channels), branches and invariants, which cannot be represented by types alone, cf. [20]. The enriched expressiveness of specifications introduces technical challenges: in particular, consistency of specifications becomes non-trivial. The presented consistency condition for global assertions is mechanically checkable relatively to the decidability of the underling logic, and ensures that the end-point assertions are automatically consistent when projected. On this basis a sound and relatively complete proof system is built that guarantees semantic consistency.

As a different DbC-based approach to concurrency, an extension of DbC has been proposed in [27], using contracts for SCOOP [26] in order to reason about liveness properties of concurrent object-oriented systems. The main difference of our approach from [27] is that our framework specifies focuses on systems based on distributed message passing systems while [27] treats shared resources. The notion of pre-/post-conditions and invariants for global assertions centring on communications and the use of projections are not found in [27]. The treatment of liveness in our framework is an interesting topic for further study.

References

1. Acciai, L., Borale, M.: A type system for client progress in a service-oriented calculus. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) Concurrency, Graphs and Models. LNCS, vol. 5065, pp. 625–641. Springer, Heidelberg (2008)
2. Berger, M., Honda, K., Yoshida, N.: Completeness and logical full abstraction for modal logics for the typed π-calculus. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 99–111. Springer, Heidelberg (2008)
3. Bettini, L., et al.: Global Progress in Dynamically Interfered Multiparty Sessions. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 418–433. Springer, Heidelberg (2008)
4. Bhargavan, K., Corin, R., Deniélou, P.M., Fournet, C., Leifer, J.: Cryptographic protocol synthesis and verification for multiparty sessions. In: CSF, pp. 124–140 (2009)
5. Bhargavan, K., Fournet, C., Gordon, A.D.: Modular verification of security protocol code by typing. In: POPL, pp. 445–456 (2010)
6. Bonelli, E., Compagnoni, A., Gunter, E.: Correspondence assertions for process synchronization in concurrent communications. JFC 15(2), 219–247 (2005)

7. Bravetti, M., Zavattaro, G.: A foundational theory of contracts for multi-party service composition. Fundamenta Informaticae XX, 1–28 (2008)
8. Buscemi, M., Montanari, U.: CC-Pi: A constraint-based language for specifying service level agreements. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 18–32. Springer, Heidelberg (2007)
9. Caires, L., Vieira, H.T.: Conversation types. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 285–300. Springer, Heidelberg (2009)
10. Castagna, G., Padovani, L.: Contracts for mobile processes. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 211–228. Springer, Heidelberg (2009)
11. Dam, M.: Proof systems for pi-calculus logics. In: Logic for Concurrency and Synchronisation. Trends in Logic, Studia Logica Library, pp. 145–212. Kluwer, Dordrecht (2003)
12. De Nicola, R., et al.: A Basic Calculus for Modelling Service Level Agreements. In: Coordination. LNCS, vol. 3454, pp. 33–48. Springer, Heidelberg (2005)
13. Floyd, R.W.: Assigning meaning to programs. In: Proc. Symp. in Applied Mathematics, vol. 19 (1967)
14. Frankel, D.S.: Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley, Chichester (2003)
15. Freeman, T., Pfenning, F.: Refinement types for ml. SIGPLAN Not. 26(6), 268–277 (1991)
16. Guttman, J.D., et al.: Trust management in strand spaces: A rely-guarantee method. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 325–339. Springer, Heidelberg (2004)
17. Hennessy, M., Milner, R.: Algebraic laws for non-determinism and concurrency. JACM 32(1) (1985)
18. Hoare, T.: An axiomatic basis of computer programming. CACM 12 (1969)
19. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type disciplines for structured communication-based programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 22–138. Springer, Heidelberg (1998)
20. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: POPL, pp. 273–284. ACM, New York (2008)
21. The Java Modeling Language (JML) Home Page
22. Jones, C.B.: Specification and design of (parallel) programs. In: IFIP Congress, pp. 321–332 (1983)
23. Leino, K.R.M.: Verifying object-oriented software: Lessons and challenges. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, p. 2. Springer, Heidelberg (2007)
24. Mendelson, E.: Introduction to Mathematical Logic. Wadsworth Inc., Bermont (1987)
25. Meyer, B.: Applying "Design by Contract". Computer 25(10), 40–51 (1992)
26. Meyer, B.: Object-Oriented Software Construction, ch. 31. Prentice Hall, Englewood Cliffs (1997)
27. Nienaltowski, P., Meyer, B., Ostroff, J.S.: Contracts for concurrency. Form. Asp. Comput. 21(4), 305–318 (2009)
28. OMG: Object Constraint Language Version 2.0 (May 2006)
29. Peyton Jones, S., et al.: Composing contracts: an adventure in financial engineering. In: ICFP, pp. 281–292. ACM, New York (2000)
30. Pierce, B.C.: Types and Programming Languages. MIT Press, Cambridge (2002)
31. Full version of this paper, http://www.cs.le.ac.uk/people/lb148/fullpaper.html
32. SAVARA JBoss Project webpage, http://www.jboss.org/savara
33. Vallecillo, A., Vasconcelos, V.T., Ravara, A.: Typing the behavior of objects and components using session types. Fundamenta Informaticæ 73(4), 583–598 (2006)
34. Xi, H., Pfenning, F.: Dependent types in practical programming. In: POPL, pp. 214–227. ACM, New York (1999)
35. Xu, D., Peyton Jones, S.: Static contract checking for Haskell. In: POPL, pp. 41–52. ACM, New York (2009)

Bisimilarity of One-Counter Processes Is PSPACE-Complete

Stanislav Böhm[1,⋆], Stefan Göller[2], and Petr Jančar[1]

[1] Techn. Univ. Ostrava (FEI VŠB-TUO), Dept of Computer Science, Czech Republic
[2] Universität Bremen, Institut für Informatik, Germany

Abstract. A one-counter automaton is a pushdown automaton over a singleton stack alphabet. We prove that the bisimilarity of processes generated by nondeterministic one-counter automata (with no ε-steps) is in PSPACE. This improves the previously known decidability result (Jančar 2000), and matches the known PSPACE lower bound (Srba 2009). We add the PTIME-completeness result for deciding regularity (i.e. finiteness up to bisimilarity) of one-counter processes.

1 Introduction

Among the various notions of behavioral equivalences of (reactive) systems, (strong) *bisimilarity* plays an important rôle (cf, e.g., [[16]]). For instance, various logics can be characterized as the bisimulation-invariant fragment of richer logics. A famous theorem due to van Benthem states that the properties expressible in modal logic coincide with the bisimulation-invariant properties expressible in first-order logic [28]. Similar such characterizations have been obtained for the modal μ-calculus [8] and for CTL* [17]. Another important notion is *weak bisimilarity* that generalizes (strong) bisimilarity by distinguishing ε-moves corresponding to internal behavior. There are numerous further notions of equivalences. For a more detailed treatment of the different behavioral equivalences in the context of concurrency theory, the reader is referred to [4].

The *(weak/strong) bisimilarity problem* consists in deciding if two given states of a given transition system are weakly/strongly bisimilar. On *finite transition systems* both weak and strong bisimilarity is well-known to be complete for deterministic polynomial time [1]. Moreover, on finite transition systems weak bisimilarity can be reduced to strong bisimilarity in polynomial time by computing the transitive closure.

In the last twenty years a lot of research has been devoted to checking behavioral equivalence of infinite-state systems, see [23] for an up-to-date record. In the setting of infinite-state systems, see also [14] for Mayr's classification of infinite-state systems, the situation is less clear. There are numerous classes of infinite-state systems for which *decidability* of bisimilarity is not known. Three such intricate open problems are (i) weak bisimilarity on basic parallel processes (BPP, a subclass of Petri nets), (ii) strong bisimilarity of process algebras (PA), and (iii) weak bisimilarity of basic process algebras (BPA). On the negative side, we mention undecidability of weak bisimilarity of PA by Srba [22]. On the positive side we mention an important result by Sénizergues who

⋆ S. Böhm and P. Jančar are supported by the Czech Ministry of Education, project No. 1M0567.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 177–191, 2010.

shows that bisimilarity on equational graphs of finite out degree [19] (a slight generalization of pushdown graphs) is decidable. See also Stirling's unpublished paper [25] for a shorter proof of this, using ideas from concurrency theory. For normed PA processes Hirshfeld and Jerrum prove decidability of strong bisimilarity [7].

When focussing on the *computational complexity* of bisimilarity checking of infinite-state systems for which this problem is decidable, the situation becomes even worse. There are only very few classes of infinite-state systemss for which the precise computational complexity is known. For instance, when coming back to one of the above-mentioned positive results by Sénizergues/Stirling concerning (slight extensions of) pushdown graphs, a primitive recursive upper bound is not yet known. However, EXPTIME hardness of this problem was proven by Kučera and Mayr [13]. As one of the few results on infinite systems where the upper and lower complexity bounds match, we can mention [10] where it is shown that bisimilarity on basic parallel processes is PSPACE-complete.

In this paper we study the computational complexity of deciding strong bisimilarity over processes generated by one-counter automata. One-counter automata are pushdown automata over a singleton stack alphabet. This model has been extensively studied in the verification community; we can name, e.g., [2,5,6,3,26] as recent works. Weak bisimilarity for one-counter processes is shown to be undecidable in [15], via a reduction from the emptiness problem of Minsky machines.

For strong bisimilarity the third author established decidability in [9], however without providing any precise complexity bounds. In an unpublished article [29] Yen analyses the approach of [9], deriving a triply exponential space upper bound. A PSPACE lower bound for bisimilarity is proven by Srba [24]. This lower bound already holds over one-counter automata that cannot test for zero and whose actions can moreover be restricted to be *visible* (so called *visibly one-counter nets*), i.e. that the label of the action determines if the counter is incremented, decremented, or not modified respectively. For visibly one-counter automata it is proven in [24] that strong bisimilarity is in PSPACE via reduction to the model checking problem of the modal μ-calculus over one-counter processes [20]. For bisimilarity on general one-counter processes, in particular when dropping the visibility restriction, the situation is surely more involved.

Our main result closes the complexity gap for bisimilarity of one-counter processes from above, thus establishing PSPACE-completeness. In a nutshell, we provide a nondeterministic algorithm implementable in polynomial space which generates a bisimulation relation *on-the-fly*. The algorithm uses a polynomial-time procedure which, given a pair $p(m), q(n)$ of processes, either gives a definite answer 'surely bisimilar' or 'surely non-bisimilar', or declares the pair as a *candidate*. For each fixed m there are (only) polynomially many candidates $(p(m), q(n))$, and the algorithm processes each $m = 0, 1, 2, \ldots$ in turn, guessing the bisimilarity status of all respective candidates and verifying the (local) consistency of the guesses. A crucial point is that it is sufficient to stop the processing after exponentially many steps, since then a certain periodicity is guaranteed, which would enable to successfully continue forever.

We also consider the problem of deciding *regularity* (finiteness w.r.t. bisimilarity) which asks if, for a given one-counter process, there is a bisimilar state in some finite system. Decidability of this problem was proven in [9] and according to [24] it follows from [1] and [21] that the problem is also hard for P. We give a simpler P-hardness proof, but we also show that the regularity problem is in P, thus establishing its P-completeness. It is appropriate to add that Kučera [12] showed a polynomial algorithm deciding bisimilarity between a one-counter process and a (given) finite system state.

The paper is organized as follows. Section 2 contains the basic notions, definitions, and recalls some auxiliary results. Section 3 recalls and enhances some useful notions which were used in [9] and elsewhere. Section 4 contains the crucial technical results, which have enabled to replace the decision algorithm from [9] with a polynomial space algorithm. The algorithm is elaborated in Section 5 and its correctness is shown in Section 6. Section 7 then shows PTIME-completeness of $\sim$-regularity.

2 Preliminaries

$\mathbb{N}$ denotes the set $\{0, 1, 2, \ldots\}$. For a set X, by $|X|$ we denote its cardinality.

Transition systems. A *(labelled) transition system* is a structure $T = (S, \mathsf{A}, \{\xrightarrow{a} \mid a \in \mathsf{A}\})$, where S is a set of *states*, A a set of *actions*, and $\xrightarrow{a} \subseteq S \times S$ is a set of a-labeled *transitions*, for each action $a \in \mathsf{A}$. We define $\longrightarrow = \bigcup_{a \in \mathsf{A}} \xrightarrow{a}$, and prefer to use the infix notation $s_1 \xrightarrow{a} s_2$ (resp. $s_1 \longrightarrow s_2$) instead of $(s_1, s_2) \in \xrightarrow{a}$ (resp. $(s_1, s_2) \in \longrightarrow$). T is a *finite transition system* if S and A are finite; we then define the *size* of T as $|T| = |S| + |\mathsf{A}| + \sum_{a \in \mathsf{A}} |\xrightarrow{a}|$.

Bisimulation equivalence. Let $T = (S, \mathsf{A}, \{\xrightarrow{a} \mid a \in \mathsf{A}\})$ be a transition system. A binary relation $R \subseteq S \times S$ is a *bisimulation* if for each $(s_1, s_2) \in R$ the following *bisimulation condition* holds:

- for each $s_1' \in S$, $a \in \mathsf{A}$, where $s_1 \xrightarrow{a} s_1'$, there is some $s_2' \in S$ such that $s_2 \xrightarrow{a} s_2'$ and $(s_1', s_2') \in R$, and
- for each $s_2' \in S$, $a \in \mathsf{A}$, where $s_2 \xrightarrow{a} s_2'$, there is some $s_1' \in S$ such that $s_1 \xrightarrow{a} s_1'$ and $(s_1', s_2') \in R$.

We say that states s_1 and s_2 are *bisimilar*, abbreviated by $s_1 \sim s_2$, if there is a bisimulation R containing (s_1, s_2). Bisimilarity $\sim$ is obviously an equivalence. We also note that the union of bisimulations is a bisimulation, and that $\sim$ is the maximal bisimulation on S. Bisimilarity is naturally defined also between states of different transition systems (by considering their disjoint union).

One-counter automata. A *one-counter automaton* is a tuple $M = (Q, \mathsf{A}, \delta_{=0}, \delta_{>0})$, where Q is a finite nonempty set of *control states*, A is a finite set of *actions*, $\delta_{=0} \subseteq Q \times \{0, 1\} \times \mathsf{A} \times Q$ is a finite set of *zero transitions*, and $\delta_{>0} \subseteq Q \times \{-1, 0, 1\} \times \mathsf{A} \times Q$ is a finite set of *positive transitions*. (There are no ε-steps in M.)

The *size* of M is defined as $|M| = |Q| + |\mathsf{A}| + |\delta_{=0}| + |\delta_{>0}|$. Each one-counter automaton $M = (Q, \mathsf{A}, \delta_{=0}, \delta_{>0})$ defines the transition system $T_M = (Q \times \mathbb{N}, \mathsf{A}, \{\xrightarrow{a} \mid a \in \mathsf{A}\})$, where $(q, n) \xrightarrow{a} (q', n + i)$ iff either $n = 0$ and $(q, i, a, q') \in \delta_{=0}$, or $n > 0$ and $(q, i, a, q') \in \delta_{>0}$.

A *one-counter net* is a one-counter automaton, where $\delta_{=0} \subseteq \delta_{>0}$.

A state (q, m) of T_M is also called a *configuration* of M, or a *one-counter process*; we usually write it as $q(m)$. Elements of $\delta_{=0} \cup \delta_{>0}$ are called *transitions*. The notion of a *path* $p(m) \xrightarrow{\sigma} q(n)$, where σ is a *sequence of transitions*, is defined in the natural way. A transition sequence β in $(\delta_{>0})^+$ is called an *elementary cycle* if it induces an elementary cycle in the control state set Q, i.e., if $\beta = (q_1, i_1, a_1, q_2), (q_2, i_2, a_2, q_3), \ldots (q_m, i_m, a_m, q_{m+1})$ where $q_i \neq q_j$ for $1 \leq i < j \leq m$ and $q_{m+1} = q_1$. We note that such a cycle has length at most $|Q|$, and its effect (i.e., the caused change) on the counter value is in the set $\{-|Q|, -|Q|+1, \ldots, |Q|\}$.

Decision problems. We are interested in the following two decision problems.

BISIMILARITY ON OCA

INPUT: A one-counter automaton M with two states $p_0(m_0)$ and $q_0(n_0)$ of T_M, where both m_0 and n_0 are given in binary.
QUESTION: Is $p_0(m_0) \sim q_0(n_0)$?

We say that a *one-counter process* $q(n)$, i.e. a configuration $q(n)$ of a one-counter automaton M, is *$\sim$-regular* (or *finite up to bisimilarity*) if there is a finite transition system with some state s such that $q(n) \sim s$.

$\sim$REGULARITY ON OCA

INPUT: A one-counter automaton M and a state $q(n)$ of T_M (n given in binary).
QUESTION: Is $q(n)$ $\sim$-regular?

Stratified bisimilarity. Given a transition system $T = (S, \mathsf{A}, \{\xrightarrow{a} \mid a \in \mathsf{A}\})$, on S we define the family of i-equivalences, $i \in \mathbb{N}$, $\sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \cdots$ as follows. We put $\sim_0 = S \times S$, and we have $s_1 \sim_{i+1} s_2$ if the following two conditions hold:

- for each $s'_1 \in S$, $a \in \mathsf{A}$, where $s_1 \xrightarrow{a} s'_1$, there is some $s'_2 \in S$ such that $s_2 \xrightarrow{a} s'_2$ and $s'_1 \sim_i s'_2$;
- for each $s'_2 \in S$, $a \in \mathsf{A}$, where $s_2 \xrightarrow{a} s'_2$, there is some $s'_1 \in S$ such that $s_1 \xrightarrow{a} s'_1$ and $s'_1 \sim_i s'_2$.

The following proposition is an instance of the result for image finite systems [16].

Proposition 1. *On states of T_M we have $\sim = \bigcap_{i \geq 0} \sim_i$.*

2.1 Some Useful Observations

The next proposition captures the locality of the bisimulation condition for one-counter automata, implied by the fact that the counter value can change by at most 1 in a move.

Proposition 2. *Given a one-counter automaton $M = (Q, \mathsf{A}, \delta_{=0}, \delta_{>0})$ and a relation $\mathcal{R} \subseteq (Q \times \mathbb{N}) \times (Q \times \mathbb{N})$, for checking if a pair $(p(m), q(n)) \in \mathcal{R}$ satisfies the bisimulation condition it suffices to know the restriction of $\mathcal{R}$ to the set* $\text{NEIGHBOURS}(m, n) = \{\, (p'(m'), q'(n')) \mid |m' - m| \leq 1, |n' - n| \leq 1 \,\}$.

Standard partition arguments [11,18] imply the following proposition for *finite* systems.

Proposition 3. *Given a finite transition system $F = (Q, \mathsf{A}, \{\xrightarrow{a} \mid a \in \mathsf{A}\})$, where $k = |Q|$, we have $\sim_{k-1} = \sim_k = \sim$ on Q. Moreover, (the partition of Q corresponding to) $\sim$ can be computed in polynomial time.*

3 Underlying Finite Automaton and the Set INC

Most of the notions, claims and ideas in this section appeared in [9] and elsewhere; nevertheless, we present (and extend) them in a concise self-contained way.

If the context does not indicate otherwise, in what follows we (often implicitly) assume a fixed one-counter automaton $M = (Q, \mathsf{A}, \delta_{=0}, \delta_{>0})$, using k for $|Q|$. We start by observing that if the counter value is large, then M behaves, for a long time, like a (nondeterministic) finite automaton. By F_M we denote the *finite transition system underlying* M; we put $F_M = (Q, \mathsf{A}, \{\xrightarrow{a} \mid a \in \mathsf{A}\})$, where $\xrightarrow{a} = \{(q_1, q_2) \in Q \times Q \mid \exists i : (q, i, a, q') \in \delta_{>0}\}$. ($F_M$ thus behaves as if the counter is positive, ignoring the counter changes.) In what follows, $p, q, r \in Q$ are viewed as control states of M or as states of F_M, depending on context. Our observation is formalized by the next proposition (which is obvious, by induction on m).

Proposition 4. *If $m' \geq m$ then $p(m') \sim_m p$.*
(Here $p(m')$ is a state of T_M, whereas p is a state of F_M.)

This implies, e.g., that if $p \not\sim q$ (i.e., $p \not\sim_k q$ by Proposition 3) and $m, n \geq k$, then $p(m) \not\sim_k q(n)$ (and thus $p(m) \not\sim q(n)$), since $p(m) \sim_k p$, $q(n) \sim_k q$ and $\sim_k$ is an equivalence. If $p \sim q$ then we can have $p(m) \not\sim q(n)$, due to the possibility of reaching zero. For making this more precise, we define the following set

$$\text{INC} = \{\, r(\ell) \mid \forall q \in Q : r(\ell) \not\sim_k q \,\}\,.$$

The configurations in INC are *incompatible with* F_M in the sense that they are not bisimilar up to k moves with any state of F_M.

Proposition 5. *If $r(\ell) \in$ INC then $\ell < k$. Moreover,* INC *can be constructed in polynomial time.*

Proof. If $\ell \geq k$ then $r(\ell) \sim_k r$, and thus $r(\ell) \notin$ INC. To construct INC, we can start with the set containing all k^3 pairs $(r(\ell), q)$, where $\ell < k$; all such pairs belong to $\sim_0$. We then delete the pairs not belonging to $\sim_1$, then those not belonging to $\sim_2$, etc., until $\sim_k$. The configurations $r(\ell)$ for which no pair $(r(\ell), q)$ survived, are in INC. (This process can be done simultaneously for the pairs (p, q) in F_M; we also use the fact $r(k) \sim_k r$.) □

The arguments of the previous proof also induce the following useful proposition.

Proposition 6. *The question if $p(m) \sim_k q(n)$ can be decided in polynomial time.*

We note that if $p(m) \in$ INC and $q(n) \notin$ INC then $p(m) \not\sim q(n)$ (in fact, $p(m) \not\sim_k q(n)$). More generally, if two one-counter processes are bisimilar then they must agree on the distance to INC; this is formalized by the next lemma. We define

$$\text{dist}(p(m))$$

as the length of the shortest transition sequence σ such that $p(m) \xrightarrow{\sigma}$ INC (i.e., $p(m) \xrightarrow{\sigma} r(\ell)$ for some $r(\ell) \in$ INC); we put dist$(p(m)) = \omega$ if there is no such sequence, i.e., when INC is unreachable, denoted $p(m) \not\rightarrow^{*}$ INC.

Lemma 7. *If $p(m) \sim q(n)$ then dist$(p(m)) =$ dist$(q(n))$.*

Proof. For the sake of contradiction, suppose that $p(m) \sim q(n)$ and $d = \text{dist}(p(m)) < \text{dist}(q(n))$, for the least d; necessarily $d > 0$, since we cannot have $p(m) \in$ INC, $q(n) \notin$ INC. Thus there is a move $p(m) \xrightarrow{a} p'(m')$ with dist$(p'(m')) = d{-}1$, which must be matched by some $q(n) \xrightarrow{a} q'(n')$ where $p'(m') \sim q'(n')$. Necessarily $d{-}1 = \text{dist}(p'(m')) < \text{dist}(q'(n'))$, which contradicts the minimality of d. □

The next lemma clarifies the opposite direction in the case of infinite distances.

Lemma 8. *If dist$(p(m)) = \omega$ then $p(m) \sim r$ for some $r \in Q$. Thus if dist$(p(m)) =$ dist$(q(n)) = \omega$ then $p(m) \sim q(n)$ iff there is some $r \in Q$ such that $p(m) \sim_k r \sim_k q(n)$.*

Proof. If dist$(p(m)) = \omega$, i.e. $p(m) \not\rightarrow^{*}$ INC, then in particular $p(m) \notin$ INC, and there is thus $r \in Q$ such that $p(m) \sim_k r$. We can easily check that

$$R = \{\, (p(m), r) \mid p(m) \sim_k r,\, p(m) \not\rightarrow^{*} \text{INC}\,\}$$

is a bisimulation: if $p(m) \xrightarrow{a} p'(m')$ and $r \xrightarrow{a} r'$ where $p'(m') \sim_{k-1} r'$, then $p'(m') \not\rightarrow^{*}$ INC and the fact $p'(m') \notin$ INC implies that $r'' \sim_k p'(m') \sim_{k-1} r'$ for some $r'' \in Q$; hence $r' \sim_{k-1} r''$ and thus $r' \sim_k r''$ (by Proposition 3), which means $p'(m') \sim_k r'$. □

In the next section we look in more detail at the function $dist(p(m))$, which provides a useful constraint on bisimilar pairs. But before that, we partition the set $(Q \times \mathbb{N}) \times (Q \times \mathbb{N})$ into three categories. We say that a pair $(p(m), q(n))$ is

- *surely-positive* if $dist(p(m)) = dist(q(n)) = \omega$ and $p(m) \sim_k q(n)$ (and thus surely $p(m) \sim q(n)$, by Lemma 8),
- *surely-negative* if $p(m) \not\sim_k q(n)$ or $dist(p(m)) \neq dist(q(n))$ (and thus surely $p(m) \not\sim q(n)$),
- *candidate* otherwise, i.e., if $p(m) \sim_k q(n)$ and $dist(p(m)) = dist(q(n)) < \omega$.

By SUREPOS we denote the set of all surely-positive pairs, and we note the following obvious proposition.

Proposition 9. SUREPOS *is a bisimulation.*

It will be also useful to view the set CAND of all candidate pairs as the union

$$\text{CAND} = \text{CAND}_0 \cup \text{CAND}_1 \cup \text{CAND}_2 \cup \cdots$$

where CAND$_i$ contains the *candidate pairs at level i*, i.e. the pairs $(p(m), q(n)) \in$ CAND with $m = i$.

4 Distance to INC

In this section we look at the distance function $dist(p(m))$ in more detail (Lemma 10) and derive some consequences (Lemma 11) which will be useful for the design and analysis of our later algorithm.

We start with sketching some intuition which is then formalized in Lemma 10. To reach INC from $p(m)$ most quickly, for a large m, one uses a suitable prefix arriving at a 'most effective' elementary cycle $q(-) \xrightarrow{\beta} q(-)$ (which decreases the counter by $k = |Q|$ at most), let us call it a *d-cycle*, then repeats the d-cycle sufficiently many times, and finishes with a suffix arriving at INC. It is not difficult to anticipate that one can bound the length (and thus also the counter change) of the prefix and the suffix by a (small degree) polynomial $\mathrm{pol}(k)$. We now state the lemma. For technical reasons, we do not require explicitly that the d-cycle is elementary; it is sufficient to bound its length by k.

Lemma 10. *There is a polynomial pol* : $\mathbb{N} \to \mathbb{N}$ *(independent of M) such that for any $p(m)$ with $dist(p(m)) < \omega$ there is a shortest path $p(m) \xrightarrow{\sigma}$ INC with the transition sequence σ of the form $\sigma = \alpha\beta^i\gamma$ where $length(\alpha\gamma) \leq pol(k)$ and β is a decreasing cycle of length $\leq k$.*

Proof. To give a complete formal proof requires some technical work. Since the essence of the claim is not original and similar technical results appear in the previous works on one-counter automata, we do not provide a self-contained proof, but we use Lemma 2 from an older paper [27]; in our notation, this lemma is formulated as follows:

> **Claim.** If there is a positive path (using positive transitions) from $p(m)$ to $q(n)$ and $m - n \geq k^2$ and $n \geq k^2$ then there is a shortest path $p(m) \xrightarrow{\sigma} q(n)$ such that $\sigma = \alpha\beta^i\gamma$ where $length(\alpha\gamma) < k^2$ and $length(\beta) \leq k$.

(Although [27] studies *deterministic* one-counter automata, the lemma obviously applies to our nondeterministic case as well, since we can view the transitions themselves as the actions.) We note that if $m - n \geq k^2 + k$ then β is necessarily a decreasing cycle ($i \geq 2$ in this case). It is also clear that the (shortest) path $p(m) \xrightarrow{\sigma} q(n)$ in the Claim does not visit any $q'(n')$ with $n' \geq m + k^2 + k$; we say that the path moves in the "$< (m+k^2+k)$ area" (note that the prefix α moves in the "$< (m+k^2)$ area" and the suffix γ moves in the "$< (n+k^2)$ area").

Recalling that $\ell < k$ for each $r(\ell) \in$ INC, we note that any shortest path $p(m) \xrightarrow{\sigma}$ INC either moves in the "$< k^2$-area", in which case its length is bounded by k^3 (since no configuration is visited twice), or can be presented in the form $p(m) \xrightarrow{\sigma_1} q_1(k^2) \xrightarrow{\sigma_2} q_2(k^2) \xrightarrow{\sigma_3} \cdots \xrightarrow{\sigma_m} q_m(k^2) \xrightarrow{\sigma_{m+1}}$ INC where $1 \leq m \leq k$ and $q_1(k^2), q_2(k^2), \ldots, q_m(k^2)$ are all configurations on the path which have the counter value k^2. By the above considerations, the segment $q_i(k^2) \xrightarrow{\sigma_{i+1}}$ moves in the "$< (3k^2 + k)$ area", and its length is thus bounded by $k \cdot (3k^2+k)$. The segment $p(m) \xrightarrow{\sigma_1} q_1(k^2)$ either moves in the "$< 2k^2$ area", in which case its length is bounded by $2k^3$, or it can be written $p(m) \xrightarrow{\sigma_1'} p'(m') \xrightarrow{\sigma_2'} q_1(k^2)$ where $m' \geq 2k^2$ and σ_1' (which might be empty) is bounded by $2k^3$. The statement of our Lemma thus follows from the above Claim applied to the segment $p'(m') \xrightarrow{\sigma_2'} q_1(k^2)$. □

The next lemma lists some important consequences. A main point is to clarify the distribution of the set CAND. Informally speaking, the candidate pairs are contained inside polynomially many linear belts, each belt having a rational slope, being a fraction of polynomially bounded integers, as well as a polynomially bounded (vertical) thickness.

Remark. It is helpful to think in geometrical notions. Every relation $\mathcal{R} \subseteq (Q \times \mathbb{N}) \times (Q \times \mathbb{N})$ can be viewed as a 'coloring' $\chi_{\mathcal{R}} : Q \times Q \times \mathbb{N} \times \mathbb{N} \to \{\bullet, \circ\}$; for each $p, q \in Q$ it prescribes a black-white coloring of the plane (grid) $\mathbb{N} \times \mathbb{N}$. This was more formalized in [9]; here we just informally use Figure 1.

Lemma 11

1. *There is a polynomial-time algorithm computing $dist(p(m))$ for any p, m; here the size of the input is $|M| + \log m$ (m is written in binary).*
2. *If $dist(p(m)) < \omega$ then*

$$dist(p(m)) = \frac{c_1}{c_2}(m + d_1) + d_2 = \frac{c_1}{c_2}m + \psi$$

for some integers $0 \leq c_1 \leq k$, $1 \leq c_2 \leq k$, $|d_1| \leq pol_1(k)$, $0 \leq d_2 \leq pol_1(k)$ where pol_1 is a polynomial (independent of M); the values c_1, c_2, d_1, d_2 generally depend on p, m.
Moreover, for the rational number $\psi = \frac{c_1}{c_2}d_1 + d_2$ we have $|\psi| \leq (k+1) \cdot pol_1(k)$.

3. *If $dist(p(m)) = dist(q(n)) < \omega$ then*

$$n = \rho \cdot m + \xi$$

where (the slope) ρ is either 0 or of the form $\frac{c_1 c_2'}{c_2 c_1'}$, for $c_1, c_2, c_1', c_2' \in \{1, 2, \ldots, k\}$, and $|\xi|$ is bounded by a polynomial $pol_2(k)$.
(This formalizes the above announced polynomially many belts, with the vertical thickness $1 + 2 \cdot pol_2(k)$.)

4. *There is a polynomial pol_4 such that for each $m \geq pol_4(k)$ we have $\rho_1 \cdot m + pol_2(k) + 1 < \rho_2 \cdot (m-1) - pol_2(k)$, where $\rho_1 < \rho_2$ are (different) slopes from Point 3, pol_2 also being taken from there.*
(I.e., for levels $m \geq pol_4(k)$ the belts are separated, in the sense that no two pairs from different belts are neighbours.)

5. *There is a polynomial-time algorithm which, given i (in binary), computes the set* CAND_i *of all* candidate pairs at level *i (all pairs $(p(i), q(n))$ such that $p(i) \sim_k q(n)$ and $dist(p(i)) = dist(q(n)) < \omega$). We have $|\mathrm{CAND}_i| \leq pol_3(k)$ for a polynomial pol_3.*

6. *If Δ is a multiple of the effects of all decreasing cycles of length $\leq k$ (the absolute values of the effects are in the set $\{1, 2, \ldots, k\}$) then for each $m \geq k + pol(k)$, where pol is taken from Lemma 10, we have:*

$$p(m) \to^* \mathrm{INC} \;\; \textit{iff} \;\; p(m + \Delta) \to^* \mathrm{INC}.$$

7. *If* $m, n \geq k + pol(k)$ *then*

$$(p(m), q(n)) \in \textsc{SurePos} \Leftrightarrow \forall i, j \in \mathbb{N} : (p(m + i\Delta), q(n + j\Delta)) \in \textsc{SurePos}$$

(where pol and Δ are as in Point 6).

Proof Point 1. By Lemma 10 we know that a shortest path $p(m) \xrightarrow{\sigma}$ INC (if there is any) is of the form

$$p(m) \xrightarrow{\alpha} q(m{+}e_1) \xrightarrow{\beta} q(m{+}e_1{-}c_2) \xrightarrow{\beta} q(m{+}e_1{-}2c_2) \xrightarrow{\beta} \cdots$$
$$\cdots \xrightarrow{\beta} q(\ell{-}e_2{+}c_2) \xrightarrow{\beta} q(\ell{-}e_2) \xrightarrow{\gamma} r(\ell) \in \mathrm{INC}$$

where e_1 is the effect (the counter change) of the prefix α, c_2 is the absolute value of the effect of the d-cycle β, and e_2 is the effect of the suffix γ ; we put $c_1 = length(\beta)$, $c_3 = length(\alpha)$, $c_4 = length(\gamma)$. Let us recall that $0 \leq c_2 \leq c_1 \leq k$ and that the absolute values of other integers are bounded by $\mathrm{pol}(k)$ from Lemma 10.

(Independently of p, m,) we thus have polynomially many possibilities (in k) for the tuple $q, e_1, c_1, c_2, c_3, c_4, e_2, r, \ell$; these possible tuples can be processed in turn. For each tuple we can check if $(m{+}e_1){-}(\ell{-}e_2)$ is divisible by c_2 and then verify if the tuple is realizable by some appropriate α, β, γ; this verification is done by using straightforward graph reachability algorithms. (Regarding the d-cycle, it is sufficient to verify the realizability of the first segment $q(m{+}e_1) \longrightarrow q(m{+}e_1{-}c_2)$ and of the final segment $q(\ell{-}e_2{+}c_2) \longrightarrow q(\ell{-}e_2)$.) With each realizable tuple we associate the value $c_3 + c_4$ when $c_2 = 0$ and $c_3 + c_4 + \frac{c_1}{c_2}\big((m{+}e_1){-}(\ell{-}e_2)\big)$ when $c_2 > 0$. We associate ω with each non-realizable tuple. The value $dist(p(m))$ is obviously the minimal value associated with the above tuples.

Point 2. This follows immediately from the analysis in the proof of Point 1. (Since $d_2 = c_3 + c_4$, $d_1 = e_1{-}\ell{+}e_2$, it suffices to take $\mathrm{pol}_1(k) = k + \mathrm{pol}(k)$, for pol from Lemma 10. The consequence for ψ is obvious.)

Point 3. From $dist(p(m)) = \frac{c_1}{c_2}m + \psi = \frac{c_1'}{c_2'}n + \psi' = dist(q(n))$, we derive $n = \frac{c_1/c_2}{c_1'/c_2'}m + \frac{\psi-\psi'}{c_1'/c_2'}$. If $c_1 = 0$ or $c_1' = 0$ then $dist(p(m)) = dist(q(n)) \leq (k{+}1) \cdot \mathrm{pol}_1(k)$, and thus $n < k + (k{+}1) \cdot \mathrm{pol}_1(k)$ (and we can put $\rho = 0$). We can thus take $\mathrm{pol}_2(k) = 2 \cdot (k{+}1) \cdot \mathrm{pol}_1(k) \cdot k$.

Point 4. Recalling the slopes from Point 3, we note that $\rho_1 {<} \rho_2$ implies $\rho_2 \geq \rho_1 + \frac{1}{k^4}$. Since $\rho_1 \leq k^2$, it is sufficient to have $\mathrm{pol}_2(k) + 1 < \frac{1}{k^4}m - k^2 - \frac{1}{k^4} - \mathrm{pol}_2(k)$.

Point 5. Given i, for each $p \in Q$ in turn we compute $z = dist(p(i))$ and all polynomially many n such that $\frac{c_1}{c_2}(n{+}d_1){+}d_2 = z$, where c_1, c_2, d_1, d_2 satisfy the constraints from Point 2. For each such n and each $q \in Q$ we check if $(p(i), q(n)) \in \mathrm{CAND}_i$, i.e., if $dist(q(n)) = z$ and $p(i) \sim_k q(n)$; Point 1 and Proposition 6 show that this can be done in polynomial time.

Point 6. Since $m \geq k + \mathrm{pol}(k)$, the length of (each) σ such that $p(m) \xrightarrow{\sigma}$ INC is greater than $\mathrm{pol}(k)$. Increasing or decreasing the number of repeating the d-cycle does the job.

Point 7. From Point 6 we know that for $m \geq k + \mathrm{pol}(k)$ we have $dist(p(m)) = \omega$ iff $dist(p(m{+}i\Delta)) = \omega$ for all $i \in \mathbb{N}$. Thus for $m, n \geq k{+}\mathrm{pol}(k)$ we have $p(m{+}i\Delta) \sim_k q(n + j\Delta)$ and $dist(p(m + i\Delta)) = dist(q(n + j\Delta)) = \omega$ if and only if $p \sim_k q$ and $dist(p(m)) = dist(q(n)) = \omega$. □

5 A Polynomial Space Algorithm

The next lemma follows from Lemma 11, Point 1, and Proposition 6.

Lemma 12. *There is a polynomial-time algorithm which, given (M and) a pair $(p(m), q(n))$, decides if the pair is in* SUREPOS*, or in* CAND*, or is surely-negative.*

We might be tempted to try to resolve the question of bisimilarity of the candidate pairs by looking for additional polynomially checkable conditions. But the PSPACE-hardness result for (visibly) one-counter processes [24] discourages us from doing so; we should be satisfied with solving our problem in polynomial space. Thus a *nondeterministic* algorithm working in polynomial space is sufficient (since PSPACE=NPSPACE by Savitch's Theorem). We start with noting the following two obvious propositions; this will give rise to a main algorithmic idea.

Proposition 13. *For a candidate pair $(p_0(m_0), q_0(n_0)) \in \mathrm{CAND}$ we have:*
$p_0(m_0) \sim q_0(n_0)$ iff there is a subset $B \subseteq \mathrm{CAND}$ such that
$(p_0(m_0), q_0(n_0)) \in B$ and $B \cup \textsc{SurePos}$ is a bisimulation.

The following (infinite) algorithm builds a certain $B \subseteq \mathrm{CAND}$ as the union of (nondeterministically chosen) sets $B_0 \subseteq \mathrm{CAND}_0$, $B_1 \subseteq \mathrm{CAND}_1$, $B_2 \subseteq \mathrm{CAND}_2$, $\ldots$, while checking the bisimulation condition for their elements on the fly (recall the locality captured by Proposition 2). If its computation does not fail, then it is infinite and the respective set $B \subseteq \mathrm{CAND}$ (which would result as the limit) satisfies that $B \cup \textsc{SurePos}$ is a bisimulation.

- We start with putting $m = 0$, compute the set CAND_0 and (nondeterministically) choose a set $B_0 \subseteq \mathrm{CAND}_0$.
- Then we successively process $m = 0, 1, 2 \ldots$, where processing m means the following:
  - Compute CAND_{m+1} (recall Point 5 of Lemma 11) and (nondeterministically) choose $B_{m+1} \subseteq \mathrm{CAND}_{m+1}$.
  - Verify that (each pair in) B_m is (locally) correct, using B_{m-1} (when $m > 0$) and B_{m+1}, and the polynomial procedure deciding membership in SUREPOS (cf. Lemma 12).
  - (If B_m is not correct, the computation fails.)

If we force the algorithm to include the input pair $(p_0(m_0), q_0(n_0))$ into B_{m_0} then an infinite run is possible if and only if $p_0(m_0) \sim q_0(n)$. We also note that it is sufficient for the algorithm to keep only the current number m, and the sets B_{m-1} (if $m > 0$), B_m, B_{m+1} in memory. (By Point 5 of Lemma 11 this consists of at most $3 \cdot pol_3(k)$ pairs, while the bit-size of the numbers is polynomial in k and in the bit-size of m, i.e. in $\log m$.)

A final crucial point is that the algorithm, getting $p_0(m_0), q_0(n_0)$ in the input, will halt (answering $p_0(m_0) \sim q_0(n_0)$) after it has successfully processed the following levels.

$$m = 0, 1, 2, \ldots, z \text{ where } z = m_0 + \mathrm{pol}_4(k) + 2^{\mathrm{pol}_5(k)} \cdot 2^{3k \log k} \tag{1}$$

Here pol_4 is from Point 4 of Lemma 11, and we put $\text{pol}_5(k) = 2 \cdot k^2 \cdot (1 + 2 \cdot \text{pol}_2(k))$, where pol_2 is from Point 3 of Lemma 11. With this halting condition, the algorithm obviously runs in polynomial space (when given M and a pair $(p_0(m_0), q_0(n_0))$). What remains to show is the correctness of the halting condition.

6 Correctness of (The Halting Condition of) the Algorithm

Recall Points 3 and 4 of Lemma 11; the candidate pairs are contained inside polynomially many linear belts with vertical thickness $(1 + 2 \cdot \text{pol}_2(k))$, which are separated for $m \geq pol_4(k)$.

Informally speaking, if the algorithm (successfully) processes sufficiently many (exponentially many) numbers m after processing m_0, then the pigeonhole principle guarantees that a certain 'pumpable' segment appears inside each belt (this is visualized in Figure 1). At that time we are guaranteed that the relation

$$\mathcal{R} = \{(p(m), q(n)) \in B_m \cup \textsc{SurePos} \mid m \leq m_0\}$$

can be extended with certain pairs $(p'(m'), q'(n'))$, with $m' > m_0$, so that the resulting relation is a bisimulation. (These pairs $(p'(m'), q'(n'))$, $m' > m_0$, may differ from those which were actually included in $B_{m'}$ by the algorithm.) We now make this informal argument more precise.

Suppose that our algorithm successfully halts for the input pair $(p_0(m_0), q_0(n_0))$, and consider the following subsequence of the sequence (1).

$$m_0', m_0' + \Delta^3, m_0' + 2\Delta^3, m_0' + 3\Delta^3, \ldots, m_0' + 2^{\text{pol}_5(k)}\Delta^3 \tag{2}$$

where $m_0' = \max\{m_0, \text{pol}_4(k)\}$ and $\Delta = k!$; hence $\Delta \leq k^k$, and so $\Delta^3 \leq 2^{3k \log k}$.

Remark. We have chosen Δ so that Points 6 and 7 of Lemma 11 can be applied.

The chosen period Δ^3 has the following useful property. We are guaranteed that $\rho\Delta^3$ is a multiple of Δ for each slope $\rho = \frac{c_1 c_2'}{c_2 c_1'}$ $(c_1, c_2, c_1', c_2' \in \{1, 2, \ldots, k\})$ from Point 3 of Lemma 11; by Point 7 of Lemma 11 we thus also get for each $m \geq \text{pol}_4(k)$:

$$(p(m), q(n)) \in \textsc{SurePos} \Leftrightarrow \forall i \in \mathbb{N} : (p(m+i\Delta^3), q(n+i\rho\Delta^3)) \in \textsc{SurePos}\,. \tag{3}$$

(In the proof of Lemma 11, we have actually derived pol_4 satisfying $\text{pol}_4(k) \geq k{+}\text{pol}(k)$. But any polynomial pol_4 satisfying Point 4 could be replaced with a bigger one to satisfy also $\text{pol}_4(k) \geq k{+}\text{pol}(k)$ anyway.)

For a relation $\mathcal{R} \subseteq (Q \times \mathbb{N}) \times (Q \times \mathbb{N})$ and a belt, identified with its slope ρ from Point 3 of Lemma 11, we define the $\mathcal{R}$-cut of the belt ρ at level m as

$$\textsc{cut}^{\rho}_{m}(\mathcal{R}) = \{\, (p(m), q(n)) \in \mathcal{R} \mid \rho m - \text{pol}_2(k) \leq n \leq \rho m + \text{pol}_2(k) \,\}.$$

Figure 1 illustrates two cuts $\textsc{cut}^{\rho}_{m_1}(\mathcal{R})$, $\textsc{cut}^{\rho}_{m_2}(\mathcal{R})$ (the black points representing elements of $\mathcal{R}$, the white points being non-elements); the depicted cuts are 'the same' in the sense that one arises by shifting the other.

Our choice of the subsequence (2) guarantees a repeat of a '2-thick cut':

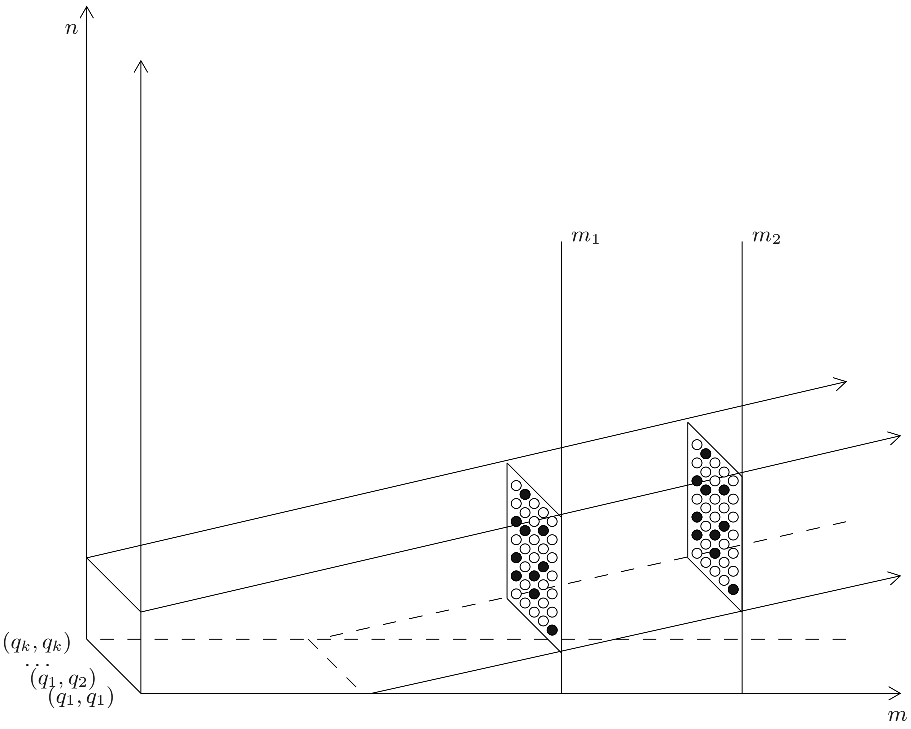


Fig. 1. Two isomorphic belt cuts in a coloring

Proposition 14. *For every $\mathcal{R}$ and 'belt' ρ there are m_1, m_2 in (2), where $m_1 < m_2$, $m_2 = m_1 + c\Delta^3$, such that*

- $(p(m_1), q(n)) \in \textsc{cut}^{\rho}_{m_1}(\mathcal{R}) \Leftrightarrow (p(m_2), q(n + \rho c\Delta^3)) \in \textsc{cut}^{\rho}_{m_2}(\mathcal{R})$,
- $(p(m_1+1), q(n)) \in \textsc{cut}^{\rho}_{m_1+1}(\mathcal{R}) \Leftrightarrow (p(m_2+1), q(n+\rho c\Delta^3)) \in \textsc{cut}^{\rho}_{m_2+1}(\mathcal{R})$.

Proof. We first note that our choice of Δ also guarantees that $\rho c\Delta^3$ is integer. Describing $\textsc{cut}^{\rho}_{m}(\mathcal{R})$ and $\textsc{cut}^{\rho}_{m+1}(\mathcal{R})$ (for any m) obviously amounts to determine a (black-point) subset of a set with (at most) $2 \cdot k^2 \cdot (1 + 2 \cdot \mathrm{pol}_2(k))$ elements; this is how we defined $\mathrm{pol}_5(k)$ in the halting condition of our algorithm (cf. (1)). There are $2^{\mathrm{pol}_5(k)}$ such subsets; thus the claim follows by the pigeonhole principle. □

Our aim is to define some relation $\mathcal{R}'$ so that $\mathcal{R}' \cup \textsc{SurePos}$ is a bisimulation and it coincides with $B \cup \textsc{SurePos}$ for the pairs $(p(m), q(n))$ with $m \leq m_0$; the set B consists of the candidate pairs included by (the successfully halting computation of) our algorithm into B_m, for $m = 0, 1, 2, \ldots, z$ as in (1).

Let us now consider a particular belt ρ. Let m_1, m_2, where $m_1 < m_2 = m_1 + c\Delta^3$, be the levels guaranteed by Proposition 14 for the relation $\mathcal{R} = B \cup \textsc{SurePos}$. Inside the belt ρ, the suggested $\mathcal{R}'$ will coincide with $\mathcal{R}$ for all levels $m \leq m_2{+}1$. For all levels $m = m_2{+}2, m_2{+}3, m_2{+}4, \ldots$, we define $\mathcal{R}'$ inside the belt ρ by the following inductive definition: for each m, n, where $m > m_2{+}1$ and $|n - \rho m| \leq \mathrm{pol}_2(k)$:

$$(p(m), q(n)) \in \mathcal{R}' \text{ iff } (p(m-c\Delta^3), q(n-\rho c\Delta^3)) \in \mathcal{R}'.$$

We note that this condition is, in fact, satisfied also for $m \in \{m_2, m_2+1\}$, due to our choice of m_1, m_2. We get the whole $\mathcal{R}'$ after having defined it inside all belts.

Proposition 15. $\mathcal{R}' \cup$ SUREPOS *is a bisimulation.*

Proof. Suppose there is a pair $(p(m), q(n)) \in \mathcal{R}' \cup$SUREPOS which does not satisfy the bisimulation condition (which is determined by the restriction to NEIGHBOURS(m, n); recall Proposition 2). We take such a pair with the least m. It is clear that $(p(m), q(n)) \notin$ SUREPOS (recall Proposition 9); moreover, the restriction of $\mathcal{R}' \cup$ SUREPOS to NEIGHBOURS(m, n) cannot be the same as for $B \cup$SUREPOS (where the algorithm verified the bisimulation condition). Hence (m, n) lies in a belt ρ, and $m \geq m_2+1$ for the respective $m_2 = m_1+c\Delta^3$. Then the pair $(p(m-c\Delta^3), q(n-\rho c\Delta^3))$ belongs to $\mathcal{R}'$ and satisfies the bisimulation condition; moreover, this pair enables the same transitions as the pair $(p(m), q(n))$. So there must be some $(p'(m'), q'(n')) \in$ NEIGHBOURS(m, n) such that $(p'(m'), q'(n')) \notin \mathcal{R}' \cup$ SUREPOS and $(p'(m'-c\Delta^3), q'(n'-\rho c\Delta^3)) \in \mathcal{R}' \cup$ SUREPOS. But this contradicts the definition of $\mathcal{R}'$ or the equivalence (3). □

Our halting condition is thus correct, and we have proved:

Theorem 16. *There is a polynomial space algorithm which, given a one-counter automaton M and a pair $p_0(m_0), q_0(n_0)$, decides if $p_0(m_0) \sim q_0(n_0)$.*

Remark. As in [9], we could derive that the bisimilarity $\sim$ (i.e., the maximal bisimulation) is 'belt-regular'. Our results here show that a natural (finite) description of this (semilinear) relation can be written in exponential space.

7 ∼-Regularity

We can easily derive the next lemma, which tells us that $p(m)$ is not $\sim$-regular iff it allows to reach states with arbitrarily large finite distances to INC.

Lemma 17. *Given $p(m)$ for a one counter automaton M, $p(m)$ is not $\sim$-regular iff for any $d \in \mathbb{N}$ there is $q(n)$ such that $p(m) \rightarrow^* q(n)$ and $d \leq dist(q(n)) < \omega$.*

The next proposition gives a more convenient characterization.

Proposition 18. *$p(m)$ is not $\sim$-regular iff $p(m) \rightarrow^* q(m+2k) \rightarrow^*$ INC for some $q \in Q$. (Recall that $k = |Q|$ for the set Q of control states of M.)*

Proof. 'Only if' is obvious.
On any path $p(m) \xrightarrow{\sigma_1} q(m+2k) \xrightarrow{\sigma_2}$ INC we have to cross the level $(m+k)$ when going up as well as when going down to INC (recall that $\ell < k$ for any $r(\ell) \in$ INC). The elementary cycles, which must necessarily appear when going up and down, can be suitably pumped to show the condition in Lemma 17. □

Lemma 19. *Deciding $\sim$-regularity of one-counter processes is in* PTIME.

Proof. We check the condition from Proposition 18. Given $p(m)$, we can compute all $q(m+2k)$ which have finite distances to INC by a polynomial algorithm (recall Point 1 of Lemma 11). When $m = 0$, the reachability of a suitable $q(2k)$ ($q(2k) \rightarrow^* \text{INC}$) can be checked straightforwardly. So we can compute all p' such that $p'(0)$ is not $\sim$-regular. Thus $p(m)$ is not $\sim$-regular iff it can reach one of the computed $q(m+2k)$ and $p'(0)$ by positive transitions. The polynomiality follows by the ideas similar to those discussed in the proof of Lemma 10. □

Lemma 20. *Deciding $\sim$-regularity (even) of one-counter nets is* PTIME*-hard.*

Proof. We use a logspace reduction from bisimilarity on finite transition systems which is PTIME-complete [1]. Given a finite transition system $(Q, \mathsf{A}, \{\xrightarrow{a}\}_{a\in\mathsf{A}})$ and $f, g \in Q$, we construct a one counter net which has the following behaviour: in $s_0(m)$, $m > 0$, it has transitions $s_0(m) \xrightarrow{a} s_0(m+1)$, $s_0(m) \xrightarrow{a} s_0(m-1)$, $s_0(m) \xrightarrow{b} f(m)$, $s_0(m) \xrightarrow{b} g(m)$. In $s_0(0)$ we only have $s_0(0) \xrightarrow{a} s_0(1)$ and $s_0(0) \xrightarrow{b} f(0)$. Any state $f(n)$ just mimicks f (not changing the counter); similarly $g(n)$ mimicks g. It is easy to verify that $s_0(n)$ is regular iff $f \sim g$. □

Theorem 21. *Deciding $\sim$-regularity of one-counter processes is* PTIME*-complete.*

Acknowledgements. We thank the reviewers for useful comments and suggestions.

References

1. Balcázar, J.L., Gabarró, J., Santha, M.: Deciding Bisimilarity is P-Complete. Formal Asp. Comput. 4(6A), 638–648 (1992)
2. Brázdil, T., Brozek, V., Etessami, K., Kučera, A., Wojtczak, D.: One-Counter Markov Decision Processes. In: Proc. of SODA, pp. 863–874. IEEE, Los Alamitos (2010)
3. Demri, S., Sangnier, A.: When Model-Checking Freeze LTL over Counter Machines Becomes Decidable. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 176–190. Springer, Heidelberg (2010)
4. Glabbeek, R.v.: The Linear Time – Branching Time Spectrum I; The Semantics of Concrete, Sequential Processes. In: Bergstra, J., Ponse, A., Smolka, S. (eds.) Handbook of Process Algebra, ch. 1, pp. 3–99. Elsevier, Amsterdam (2001)
5. Göller, S., Mayr, R., To, A.W.: On the Computational Complexity of Verifying One-Coúnter Processes. In: Proc. of LICS, pp. 235–244. IEEE Computer Society Press, Los Alamitos (2009)
6. Haase, C., Kreutzer, S., Ouaknine, J., Worrell, J.: Reachability in succinct and parametric one-counter automata. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 369–383. Springer, Heidelberg (2009)
7. Hirshfeld, Y., Jerrum, M.: Bisimulation Equivalence Is Decidable for Normed Process Algebra. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 412–421. Springer, Heidelberg (1999)
8. Janin, D., Walukiewicz, I.: On the Expressive Completeness of the Propositional mu-Calculus with Respect to Monadic Second Order Logic. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 263–277. Springer, Heidelberg (1996)
9. Jančar, P.: Decidability of Bisimilarity for One-Counter Processes. Information Computation 158(1), 1–17 (2000)

10. Jančar, P.: Strong Bisimilarity on Basic Parallel Processes is PSPACE-complete. In: Proc. of LICS, pp. 218–227. IEEE Computer Society, Los Alamitos (2003)
11. Kanellakis, P.C., Smolka, S.A.: CCS Expressions, Finite State Processes, and Three Problems of Equivalence. Information and Computation 86(1), 43–68 (1990)
12. Kučera, A.: Efficient Verification Algorithms for One-Counter Processes. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 317–328. Springer, Heidelberg (2000)
13. Kučera, A., Mayr, R.: On the Complexity of Checking Semantic Equivalences between Pushdown Processes and Finite-state Processes. Inf. Comput. 208(7), 772–796 (2010)
14. Mayr, R.: Process Rewrite Systems. Information and Computation 156(1), 264–286 (2000)
15. Mayr, R.: Undecidability of Weak Bisimulation Equivalence for 1-Counter Processes. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 570–583. Springer, Heidelberg (2003)
16. Milner, R.: Communication and Concurrency. International Series in Computer Science. Prentice Hall, Englewood Cliffs (1989)
17. Moller, F., Rabinovich, A.M.: Counting on CTL*: on the expressive power of monadic path logic. Inf. Comput. 184(1), 147–159 (2003)
18. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM Journal on Computing 16(6), 973–989 (1987)
19. Sénizergues, G.: The Bisimulation Problem for Equational Graphs of Finite Out-Degree. SIAM J. Comput. 34(5), 1025–1106 (2005)
20. Serre, O.: Parity games played on transition graphs of one-counter processes. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 337–351. Springer, Heidelberg (2006)
21. Srba, J.: Strong Bisimilarity and Regularity of Basic Process Algebra Is PSPACE-Hard. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 716–727. Springer, Heidelberg (2002)
22. Srba, J.: Undecidability of Weak Bisimilarity for PA-Processes. In: Ito, M., Toyama, M. (eds.) DLT 2002. LNCS, vol. 2450, pp. 197–208. Springer, Heidelberg (2003)
23. Srba, J.: Roadmap of Infinite results. Formal Models and Semantics, vol. 2. World Scientific Publishing Co., Singapore (2004), http://www.brics.dk/~srba/roadmap
24. Srba, J.: Beyond Language Equivalence on Visibly Pushdown Automata. Logical Methods in Computer Science 5(1:2) (2009)
25. Stirling, C.: Decidability of Bisimulation Equivalence for Pushdown Processes (2000) (unpublished manuscript)
26. To, A.W.: Model Checking FO(R) over One-Counter Processes and beyond. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 485–499. Springer, Heidelberg (2009)
27. Valiant, L.G., Paterson, M.: Deterministic one-counter automata. J. Comput. Syst. Sci. 10(3), 340–350 (1975)
28. van Benthem, J.: Modal Correspondence Theory. PhD thesis, University of Amsterdam (1976)
29. Yen, H.-C.: Complexity Analysis of Some Verification Problems for One-Counter Machines (20xx) (unpublished manuscript)

Nash Equilibria for Reachability Objectives in Multi-player Timed Games*

Patricia Bouyer, Romain Brenguier, and Nicolas Markey

LSV, ENS Cachan & CNRS, France
{bouyer,brenguie,markey}@lsv.ens-cachan.fr

Abstract. We propose a procedure for computing Nash equilibria in multi-player timed games with reachability objectives. Our procedure is based on the construction of a finite concurrent game, and on a generic characterization of Nash equilibria in (possibly infinite) concurrent games. Along the way, we use our characterization to compute Nash equilibria in finite concurrent games.

1 Introduction

Timed games. Game theory (especially games played on graphs) has been used in computer science as a powerful framework for modelling interactions in embedded systems [10, 15]. Over the last fifteen years, games have been extended with the ability to depend on timing informations. Timed games allows for a more faithful representation of reactive systems, while preserving decidability of several important properties, such as the existence of a winning strategy for one of the agents to achieve her goal, whatever the other agents do [3]. Efficient algorithms exist and have been implemented, *e.g.* in the tool UPPAAL TIGA [4].

Zero-sum vs. non-zero-sum games. In this purely antagonist view, games can be seen as two-player games, where one agent plays against another one. Moreover, the objectives of those two agents are opposite: the aim of the second player is simply to prevent the first player from winning her own objective. More generally, a (positive or negative) *payoff* can be associated with each outcome of the game, which can be seen as the amount the second player will have to pay to the first player. Those games are said to be *zero-sum*.

In many cases, however, games can be *non-zero-sum*, especially when they involve more than two agents, whose objectives may not be complementary. Such games appear *e.g.* in various problems in telecommunications, where several agents try to send data on a network [9]. Focusing only on surely-winning strategies in this setting may then be too narrow: surely-winning strategies must be winning against any behaviour of the other agents, and does not consider the fact that the other agents also try to achieve their own objectives.

Nash equilibria. In the non-zero-sum game setting, it is then more interesting to look for *equilibria*. For instance, a Nash equilibrium is a behaviour of the

* This work is partly supported by projects DOTS (ANR-06-SETI-003), QUASIMODO (FP7-ICT-STREP-214755) and GASICS (ESF-EUROCORES LogiCCC).

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 192–206, 2010.

agents in which they play rationally, in the sense that no agent can get a better payoff if she, alone, modifies her strategy [13]. This corresponds to stable states of the game. Notice that a Nash equilibrium need not exist in general, and is not necessarily optimal: several equilibria can coexist, possibly with very different payoffs.

Our contribution. We extend the study of Nash equilibria for reachability objectives (where the payoff is 1 when the objective is reached, and 0 otherwise) in the setting of timed games, as defined in [8] (but extended to n players in the obvious way). Since timed games are non-deterministic, we introduce the notion of *pseudo-Nash equilibrium*, in which non-determinism is solved "optimally" (*i.e.*, only the best outcome is considered). This corresponds to letting the players "optimally" solve non-determinism, in such a way that they have no incentive to change their choice.

As is usual in the timed setting, we rely on a region-based abstraction, which in our context is a finite concurrent game. In order to prove that the abstraction preserves Nash equilibria, we define a characterization of Nash equilibria in (possibly infinite-state) concurrent games. This characterization is built on the new concept of *repellor sets*: the repellor set for a subset A of agents is, roughly, the set of states from which players in A will not be winning in any Nash equilibrium. We explain how to compute those sets, and how they can be used to characterize Nash equilibria.

We also use repellor sets to effectively compute Nash equilibria in finite games, which solves open problems in the setting of equilibria in finite games and gives a complete solution to our original problem.

Related work. Nash equilibria (and other related solution concepts such as subgame-perfect equilibria, secure equilibria, ...) have recently been studied in the setting of (untimed) games played on a graph [6, 7, 14, 16–19]. Most of them, however, focus on turn-based games. In the setting of concurrent games, mixed strategies (*i.e.*, strategies involving probabilistic choices) are arguably more relevant than pure (*i.e.*, non-randomized) strategies. However, adding probabilities to timed strategies (over both delays and actions) involves several important technical issues (even in zero-sum non-probabilistic timed games), and we defer the study of mixed-strategy Nash equilibria in timed games to future works.

For lack of space, only sketches of proofs are given in the paper. Full proofs can be found in [5].

2 Preliminaries

We begin with defining concurrent games and Nash equilibria.

2.1 Concurrent Games

A *transition system* is a 2-tuple $\mathcal{S} = \langle \mathsf{States}, \mathsf{Edg} \rangle$ where States is a (possibly uncountable) set of states and $\mathsf{Edg} \subseteq \mathsf{States} \times \mathsf{States}$ is the set of transitions.

Given a transition system $\mathcal{S}$, a *path* π in $\mathcal{S}$ is a non-empty sequence $(s_i)_{0\leq i<n}$ (where $n \in \mathbb{N} \cup \{+\infty\}$) of states of $\mathcal{S}$ such that $(s_i, s_{i+1}) \in \mathsf{Edg}$ for all $i < n-1$. The *length* of π, denoted by $|\pi|$, is $n-1$. The set of finite paths (also called *histories* in the sequel) of $\mathcal{S}$ is denoted by[1] $\mathsf{Hist}_{\mathcal{S}}$, the set of infinite paths (also called *plays*) of $\mathcal{S}$ is denoted by $\mathsf{Play}_{\mathcal{S}}$, and $\mathsf{Path}_{\mathcal{S}} = \mathsf{Hist}_{\mathcal{S}} \cup \mathsf{Play}_{\mathcal{S}}$ is the set of paths of $\mathcal{S}$. Given a path $\pi = (s_i)_{0\leq i<n}$ and an integer $j < n$, the *j-th prefix* of π, denoted by $\pi_{\leq j}$, is the finite path $(s_i)_{0\leq i<j+1}$. If $\pi = (s_i)_{0\leq i<n}$ is a history, we write $\mathsf{last}(\pi) = s_{|\pi|}$.

We extend the definition of concurrent games given *e.g.* in [2] with non-determinism:

Definition 1. *A* non-deterministic concurrent game *is a 7-tuple* $\mathcal{G} = \langle \mathit{States}, \mathit{Edg}, \mathit{Agt}, \mathit{Act}, \mathit{Mov}, \mathit{Tab}, \preccurlyeq\rangle$ *in which:*

- $\langle \mathit{States}, \mathit{Edg}\rangle$ *is a transition system;*
- Agt *is a finite set of players (or agents);*
- Act *is a (possibly uncountable) set of actions;*
- $\mathit{Mov}\colon \mathit{States} \times \mathit{Agt} \to 2^{\mathit{Act}} \smallsetminus \{\varnothing\}$ *is a mapping indicating the actions available to a given player in a given state;*
- $\mathit{Tab}\colon \mathit{States} \times \mathit{Act}^{\mathit{Agt}} \to 2^{\mathit{Edg}} \smallsetminus \{\varnothing\}$ *associates, in a given state, a set of moves of the players with the resulting set of transitions. It is required that if $(s', s'') \in \mathit{Tab}(s, (m_A)_{A\in\mathsf{Agt}})$, then $s' = s$.*
- $\preccurlyeq\colon \mathit{Agt} \to 2^{\mathit{States}^\omega \times \mathit{States}^\omega}$ *defines, for each player, a quasi-ordering on the runs of $\mathcal{G}$, called* preference relation. *We simply write $\preccurlyeq_A$ for $\preccurlyeq(A)$.*

In the rest of this paper, we restrict to simple qualitative preference relations given by reachability conditions for each player. Formally, we assume that the preference relation is given as a tuple $(\Omega_A)_{A\in\mathsf{Agt}}$ of sets of states, and is defined as follows: if a path π visits Ω_A, then we let $v_A(\pi) = 1$, otherwise $v_A(\pi) = 0$; we then say that path π' is preferred by Player A over path π, which is written $\pi \preccurlyeq_A \pi'$, whenever either $\pi = \pi'$ or $v_A(\pi) < v_A(\pi')$.

A *deterministic* concurrent game is a concurrent game where $\mathsf{Tab}(s, (m_A)_{A\in\mathsf{Agt}})$ is a singleton for every $s \in \mathsf{States}$ and $(m_A)_{A\in\mathsf{Agt}}$ with $m_A \in \mathsf{Mov}(s, A)$. A *turn-based* game is a concurrent game for which there exists a mapping $\mathsf{Owner}\colon \mathsf{States} \to \mathsf{Agt}$ such that, for every state $s \in \mathsf{States}$, the set $\mathsf{Mov}(s, A)$ is a singleton unless $A = \mathsf{Owner}(s)$.

In a non-deterministic concurrent game, from some state s, each player A selects one action m_A among its set $\mathsf{Mov}(s, A)$ of allowed actions (the resulting tuple $(m_A)_{A\in\mathsf{Agt}}$, which we may also write m_{Agt} in the sequel, is called a *move*). This corresponds to a set of transitions $\mathsf{Tab}(s, (m_A)_{A\in\mathsf{Agt}})$, one of which is applied and gives the next state of the game. In the sequel, we abusively write $\mathsf{Hist}_{\mathcal{G}}$, $\mathsf{Play}_{\mathcal{G}}$ and $\mathsf{Path}_{\mathcal{G}}$ for the corresponding set of paths in the underlying transition system of $\mathcal{G}$. We also write $\mathsf{Hist}_{\mathcal{G}}(s)$, $\mathsf{Play}_{\mathcal{G}}(s)$ and $\mathsf{Path}_{\mathcal{G}}(s)$ for the respective subsets of paths starting in state s.

[1] For this and the coming definitions, we indicate the underlying transition system as a subscript. This may be omitted in the sequel if no ambiguity may arise.

Definition 2. *Let $\mathcal{G}$ be a concurrent game, and $A \in \mathsf{Agt}$. A* strategy *for A is a mapping $\sigma_A \colon \mathsf{Hist} \to \mathsf{Act}$ such that for any $\pi \in \mathsf{Hist}$ it holds $\sigma_A(\pi) \in \mathsf{Mov}(\mathsf{last}(\pi), A)$.*

Given a coalition (i.e., a subset of agents) $P \subseteq \mathsf{Agt}$, a strategy σ_P for coalition P is a tuple of strategies, one for each player in P. We write $\sigma_P = (\sigma_A)_{A \in P}$ for such a strategy. A strategy profile *is a strategy for the coalition Agt. We write $\mathsf{Strat}^A_{\mathcal{G}}$ for the set of strategies of player A in $\mathcal{G}$, and $\mathsf{Prof}_{\mathcal{G}}$ for the set of strategy profiles in $\mathcal{G}$.*

Notice that we only consider non-randomized (*pure*) strategies in this paper.

Let $\mathcal{G}$ be a concurrent game, P be a coalition, and σ_P be a strategy for P. A path $\pi = (s_j)_{0 \le j \le |\pi|}$ is *compatible* with the strategy σ_P if, for all $k \le |\pi| - 1$, there exists a move m_{Agt} such that:

- $m_A \in \mathsf{Mov}(s_k, A)$ for all $A \in \mathsf{Agt}$,
- $m_A = \sigma_A(\pi_{\le k})$ for all $A \in P$,
- $(s_k, s_{k+1}) \in \mathsf{Tab}(s_k, m_{\mathsf{Agt}})$.

We write $\mathsf{Out}_{\mathcal{G}}(\sigma_P)$ for the set of paths (also called *outcomes*) in $\mathcal{G}$ that are compatible with strategy σ_P of coalition P. We write $\mathsf{Out}^f_{\mathcal{G}}$ (resp. $\mathsf{Out}^\infty_{\mathcal{G}}$) for the finite (resp. infinite) outcomes, and $\mathsf{Out}_{\mathcal{G}}(s, \sigma_P)$, $\mathsf{Out}_{\mathcal{G}}(s, \sigma_P)$ and $\mathsf{Out}_{\mathcal{G}}(s, \sigma_P)$ for the respective sets of outcomes of σ_P with initial state s.

Notice that, in the case of deterministic concurrent games, any strategy profile has a single infinite outcome. This might not be the case for non-deterministic concurrent games.

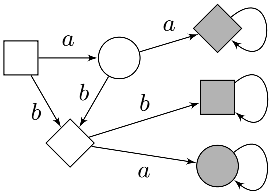


Fig. 1. A 3-player turn-based game

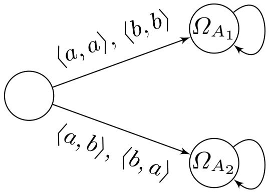


Fig. 2. A 2-player concurrent game

Example 1. Figure 1 displays an example of a three-player turn-based game. The shape of a node indicates its owner, and the goal states are those marked in grey: for instance, Player □ controls square states, and her objective is to reach ■. She cannot achieve this on her own (she has no *winning* strategy), but can achieve it with the help of Player ◇ (both should play action b).

Figure 2 is a two-player concurrent game: from the left-most state, both players choose between actions a and b, and the game goes to the top state (which is a goal state for player A_1) if they play the same action, and to the bottom state otherwise (which is a goal state for player A_2).

Given a move m_{Agt} and an action m' for some player B, we write $m_{\mathsf{Agt}}[B \mapsto m']$ for the move n_{Agt} with $n_A = m_A$ when $A \ne B$ and $n_B = m'$. This notation is extended to strategies in the natural way. In the context of non-zero-sum games, several notions of equilibria have been defined. We present here a refinement of Nash equilibria towards non-deterministic concurrent games.

Definition 3. *Let $\mathcal{G}$ be a non-deterministic concurrent game, and s be a state of $\mathcal{G}$. A* pseudo-Nash equilibrium *in $\mathcal{G}$ from s is a pair $(\sigma_{\mathsf{Agt}}, \pi)$ where $\sigma_{\mathsf{Agt}} \in \mathsf{Prof}_{\mathcal{G}}$ and $\pi \in \mathsf{Out}(s, \sigma_{\mathsf{Agt}})$ is such that for all $B \in \mathsf{Agt}$ and all $\sigma' \in \mathsf{Strat}^B$, it holds:*

$$\forall \pi' \in \mathsf{Out}(s, \sigma_{\mathsf{Agt}}[B \mapsto \sigma']).\ \pi' \preccurlyeq_B \pi.$$

Such an outcome π is called an optimal play *for the strategy profile σ_{Agt}. The* payoff *of a pseudo-Nash equilibrium $(\sigma_{\mathsf{Agt}}, \pi)$ is the function $\nu\colon \mathsf{Agt} \to \{0, 1\}$ where $\nu(A) = 1$ if π visits Ω_A (the objective of Player A), and $\nu(A) = 0$ otherwise.*

In the case of deterministic games, π is uniquely determined by σ_{Agt}, and pseudo-Nash equilibria coincide with *Nash equilibria* as defined in [13]: they are strategy profiles where no player has an incentive to unilaterally deviate from her strategy.

In the case of non-deterministic games, a strategy profile for an equilibrium may give rise to several outcomes. The choice of playing the optimal play π is then made cooperatively by all players: once a strategy profile is fixed, non-determinism is resolved by all players choosing one of the possible outcomes in such a way that each player has no incentive to unilaterally changing her choice (nor her strategy). To our knowledge, this cannot be encoded by adding an extra player for solving non-determinism. Notice that solution concepts involving an extra player for solving non-determinism can be handled by our algorithm since it yields a deterministic game (leading to *real* Nash equilibria).

Example 1 (cont'd). In the (deterministic) game of Fig. 1, the strategy profile where all players play a is *not* a Nash equilibrium from □, since player ○ would better play b and reach her winning state. The profile where they all play b is a Nash equilibrium. Actually, deterministic turn-based games such as this one always admit a Nash equilibrium [7].

Now, consider the same game as depicted in Fig. 1, but in which player ◇ has only one action available, say a, which non-deterministically leads to either ■ or ●. Then none of the two outcomes □→◇→■ and □→◇→● is globally better than the other one, hence they do not correspond to a pseudo-Nash equilibrium. The reader can check that, for any strategy profile, there never exists an optimal play, so that this modified, non-deterministic turn-based game does not admit any pseudo-Nash equilibrium.

Regarding the concurrent game of Fig. 2, it is easily seen that it also does not admit a (non-randomized) Nash equilibrium.

2.2 Decision Problems

In this paper we are interested in several decision problems related to the existence of pseudo-Nash equilibria. Let $\mathcal{S}$ be a class of concurrent games. In the sequel, we consider the following problems: given $\mathcal{G} = \langle \mathsf{States}, \mathsf{Edg}, \mathsf{Agt}, \mathsf{Act}, \mathsf{Mov},$

$\mathsf{Tab}, \preccurlyeq\rangle$ a concurrent game in class $\mathcal{S}$ with reachability objectives $\Omega_A \subseteq \mathsf{States}$ for every player A, and a state $s \in \mathsf{States}$:

Problem 1 (Existence). Does there exists a pseudo-Nash-equilibrium in $\mathcal{G}$ from s?

Problem 2 (Verification). Given a payoff $\nu\colon \mathsf{Agt} \to \{0,1\}$, does there exists a pseudo-Nash-equilibrium in $\mathcal{G}$ from s with payoff ν?

Problem 3 (Constrained-Existence). Given a constraint (given as a subset $P \subseteq \mathsf{Agt}$ and a function $\varpi\colon P \to \{0,1\}$), does there exists a pseudo-Nash-equilibrium in $\mathcal{G}$ from s with some payoff ν satisfying the constraint (*i.e.*, s.t. $\nu(A) = \varpi(A)$ for all $A \in P$)?

Notice that Problems 1 and 2 are trivially logspace-reducible to Problem 3. Together with these problems, we also consider the corresponding *function problems*: for the *verification* problem ("does the given payoff vector correspond to some equilibrium?"), the function problem asks to build a strategy profile that is an equilibrium for this payoff. For the other two problems, the function problem asks to compute a possible payoff function, and a corresponding strategy profile.

3 Qualitative Nash Equilibria

We now explain a procedure to describe pseudo-Nash equilibria in our setting. To this aim, we introduce the notion of *repellor sets*.

3.1 The Repellor Sets

Definition 4. *We define the set of* suspect *players for an edge $e = (s, s')$ given a move m_{Agt}, which we denote with $\mathsf{Susp}_{\mathcal{G}}(e, m_{\mathsf{Agt}})$, as the set:*

$$\{B \in \mathsf{Agt} \mid \exists m' \in \mathsf{Mov}(s, B) \text{ s.t. } e \in \mathsf{Tab}(s, m_{\mathsf{Agt}}[B \mapsto m'])\}.$$

We extend this notion to a finite path $\pi = (s_p)_{p \le |\pi|}$ given strategies σ_{Agt} as follows:

$$\mathsf{Susp}(\pi, \sigma_{\mathsf{Agt}}) = \bigcap_{p<|\pi|} \mathsf{Susp}((s_p, s_{p+1}), (\sigma_A(\pi_{\le p}))_{A \in \mathsf{Agt}}).$$

Intuitively, Player B is suspect for an edge e, given a move m_{Agt}, whenever she can unilaterally change her action (while the other actions are unchanged) and take edge e. Notice that if $e \in \mathsf{Tab}(s, m_{\mathsf{Agt}})$, then $\mathsf{Susp}(e, m_{\mathsf{Agt}}) = \mathsf{Agt}$. Player B is then suspect for a finite path π, given a tuple of strategies σ_{Agt}, whenever she has a strategy to enforce path π under the strategies $(\sigma_A)_{A \in \mathsf{Agt} \smallsetminus \{B\}}$ of the other players.

Lemma 5. *Given $\sigma_{\mathsf{Agt}} \in \mathsf{Prof}$ and $\pi \in \mathsf{Hist}$, the following three propositions are equivalent:*

(i) $B \in \mathsf{Susp}(\pi, \sigma_{\mathsf{Agt}})$
(ii) $\exists \sigma' \in \mathsf{Strat}^B.\ \pi \in \mathsf{Out}^f(\sigma_{\mathsf{Agt}}[B \mapsto \sigma'])$
(iii) $\pi \in \mathsf{Out}^f((\sigma_A)_{A \in \mathsf{Agt} \smallsetminus \{B\}})$

We now define the central notion of this paper, namely the *repellor sets*.

Definition 6. *Let $\mathcal{G}$ be a non-deterministic concurrent game. Given a subset $P \subseteq \mathsf{Agt}$, the* repellor set of P*, denoted by $\mathsf{Rep}_{\mathcal{G}}(P)$, is defined inductively on P as follows: as the base case, $\mathsf{Rep}_{\mathcal{G}}(\varnothing) = \mathsf{States}$; Then, assuming that $\mathsf{Rep}_{\mathcal{G}}(P')$ has been defined for all $P' \subsetneq P$, we let $\mathsf{Rep}_{\mathcal{G}}(P)$ be the largest set*[2] *satisfying the following two conditions:*

- $\forall A \in P.\ \mathsf{Rep}_{\mathcal{G}}(P) \cap \Omega_A = \varnothing$ (1)
- $\forall s \in \mathsf{Rep}_{\mathcal{G}}(P).\ \exists m_{\mathsf{Agt}} \in \mathsf{Act}^{\mathsf{Agt}}.\ \forall s' \in \mathsf{States}.$
  $$s' \in \mathsf{Rep}_{\mathcal{G}}(P \cap \mathsf{Susp}_{\mathcal{G}}((s,s'), m_{\mathsf{Agt}})) \qquad (2)$$

Intuitively, from a state in $\mathsf{Rep}_{\mathcal{G}}(P)$, the players can cooperate in order to stay in this repellor set (thus never satisfying the objectives of players in P) in such a way that breaking the cooperation does not help fulfilling one's objective.

Lemma 7. *If $P \subseteq P'$, then $\mathsf{Rep}(P') \subseteq \mathsf{Rep}(P)$.*

Remark 8. Because deterministic turn-based games are determined, they enjoy the property that $\mathsf{Rep}(\{A\}) = \mathsf{States} \smallsetminus \mathsf{Win}(\{A\})$, where $\mathsf{Win}(\{A\})$ is the set of states from which player A has a winning strategy for reaching her objective against the coalition $\mathsf{Agt} \smallsetminus \{A\}$. Notice that this does not hold in concurrent games: in the game depicted on Fig. 2, the initial state is neither in the repellor set nor in the winning set of any player.

The sets of *secure moves* for staying in $\mathsf{Rep}(P)$ is defined as:

$$\mathsf{Secure}_{\mathcal{G}}(s,P) = \{(m_i)_{A_i \in \mathsf{Agt}} \in \mathsf{Act}^{\mathsf{Agt}} \mid \forall s' \in \mathsf{States}.\ s' \in \mathsf{Rep}(P \cap \mathsf{Susp}((s,s'), m_{\mathsf{Agt}}))$$

We define the transition system $\mathcal{S}_{\mathcal{G}}(P) = (\mathsf{States}, \mathsf{Edg}')$ as follows: $(s,s') \in \mathsf{Edg}'$ iff there exists some $m_{\mathsf{Agt}} \in \mathsf{Secure}(s,P)$ such that $(s,s') \in \mathsf{Tab}(s, m_{\mathsf{Agt}})$. Note in particular that any $s \in \mathsf{Rep}(P)$ has an outgoing transition in $\mathcal{S}_{\mathcal{G}}(P)$.

Example 2. In the game of Fig. 1, state ◇ is in the repellor set of {□, ◇} and of {○, ◇} but not in that of {□, ○}. Intuitively, from that state, Player ◇ can prevent one of the other two players to reach her objective, but not both of them at the same time. It can be checked that Rep({□, ◇}) = {□; ○; ◇; ●}.

Looking now at the same game but with non-determinism in state ◇, the repellor sets are different; in particular, state ◇ is no longer in Rep({□}) nor in Rep({○}).

3.2 Using the Repellor to Characterize (pseudo-)Nash Equilibria

We now draw the link between the repellor sets and (pseudo-)Nash equilibria.

[2] This is uniquely defined since if two sets satisfy both conditions, then so does their union.

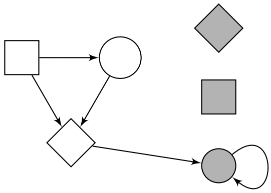


Fig. 3. $\mathcal{S}(\{\Box, \Diamond\})$ for the det. game

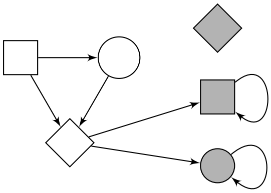


Fig. 4. $\mathcal{S}(\{\Diamond\})$ for the non-det. game

Lemma 9. *Let $P \subseteq \mathsf{Agt}$, and $s \in \mathsf{States}$. Then $s \in \mathsf{Rep}(P)$ if and only if there exists an infinite path π in $\mathcal{S}(P)$ starting from s.*

Repellor sets characterize those states from which one can find equilibria that avoid the reachability objectives of players in P:

Proposition 10. *Let $P \subseteq \mathsf{Agt}$, and $\pi \in \mathsf{Play}(s)$ be an infinite play with initial state s. Then π is a path in $\mathcal{S}(P)$ if and only if there exists $\sigma_{\mathsf{Agt}} \in \mathsf{Prof}$ such that $\pi \in \mathsf{Out}(s, \sigma_{\mathsf{Agt}})$ and for all $B \in P$ and all $\sigma' \in \mathsf{Strat}^B$ it holds:*

$$\forall \pi \in \mathsf{Out}(s, \sigma_{\mathsf{Agt}}[B \mapsto \sigma']).\ \pi \textit{ does not visit } \Omega_B.$$

Theorem 11. *Let $\mathcal{G} = \langle \mathsf{States}, \mathsf{Edg}, \mathsf{Agt}, \mathsf{Act}, \mathsf{Mov}, \mathsf{Tab}, \preccurlyeq \rangle$ be a concurrent game, with reachability objectives $\Omega_A \subseteq \mathsf{States}$ for each player $A \in \mathsf{Agt}$, and $s \in \mathsf{States}$. There is a pseudo-Nash equilibrium from s with payoff ν iff, letting P be the set $\{A \in \mathsf{Agt} \mid \nu(A) = 0\}$, there is an infinite path π in $\mathcal{S}(P)$ which starts in s and which visits Ω_A for every A not in P. Furthermore, π is the optimal play of some pseudo-Nash equilibrium.*

This gives a generic procedure to decide the existence of pseudo-Nash equilibria in non-deterministic concurrent games. It is not effective yet (remember that we allow uncountably-infinite games), but will yield algorithms when instantiated on finite games and timed games in the forthcoming sections.

Proof (of Theorem 11). ($\Rightarrow$) Let $(\sigma_{\mathsf{Agt}}, \pi)$ be a pseudo-Nash equilibrium: σ_{Agt} is a strategy profile, and $\pi \in \mathsf{Out}(s, \sigma_{\mathsf{Agt}})$ is such that for any player B and any strategy σ' for B, it holds

$$\forall \pi' \in \mathsf{Out}(s, \sigma_{\mathsf{Agt}}[B \mapsto \sigma']).\ \pi' \text{ visits } \Omega_B \Rightarrow \pi \text{ visits } \Omega_B.$$

Moreover, π visits Ω_B iff $B \notin P$. According to Proposition 10, π must be a path in the transition system $\mathcal{S}(P)$.

($\Leftarrow$) Let π be an infinite path in $\mathcal{S}(P)$ such that for every $B \notin P$, π visits some state in Ω_B. According to Proposition 10, there is a strategy profile such that π is one of its outcomes and if any player $A \in P$ deviates, no outcome visits Ω_A. Together with π, this forms a pseudo-Nash equilibrium. $\Box$

Theorem 11 gives a (necessary and sufficient) condition for the existence of a pseudo-Nash equilibrium in a game. In case an equilibrium exists, repellor sets (and the corresponding transition systems) also contain all the necessary information for effectively computing a pseudo-Nash equilibrium:

Proposition 12. *If π is an infinite path in $\mathcal{S}(P)$ from s visiting Ω_B for every $B \notin P$, then there is a pseudo-Nash equilibrium $(\sigma_{\mathsf{Agt}}, \pi)$ where strategies σ_{Agt} consist in playing secure moves in the transition system $\mathcal{S}(P \cap P')$, for some P'.*

Proof (Sketch). The strategy profile should contain π as one of its outcomes, which can be done by selecting the relevant moves from $\mathcal{S}(P)$. Now, if the play ever gets outside of π but still in $\mathsf{Rep}(P)$ (which may occur because of non-determinism, or because some player, who would have won, has deviated from her strategy), then the strategy profile should select secure moves to stay in this set. Finally, if the history exits $\mathsf{Rep}(P)$, this indicates that (at least) one player in P is trying to deviate from her selected strategy. The strategy profile must ensure that she cannot win: this is achieved by detecting the set P' of players who may be responsible for the deviation, and play secure moves in $\mathcal{S}(P \cap P')$. □

Example 3. For the game of Fig. 1, consider for instance the transition system $\mathcal{S}(\{\Box, \Diamond\})$, which is depicted on Fig. 3. There are two infinite paths from state □; they correspond to the outcomes of the two Nash equilibria in the game of Fig. 1, both of which have payoff $(\Box \mapsto 0, \bigcirc \mapsto 1, \Diamond \mapsto 0)$.

In the same game with non-determinism in state ◇, the transition system $\mathcal{S}(\{\Box, \Diamond\})$ can be checked to contain no edges, so that there is no pseudo-Nash equilibria with payoff $(\Box \mapsto 0, \bigcirc \mapsto 1, \Diamond \mapsto 0)$. Now, if we look at $\mathcal{S}(\{\Diamond\})$, which is depicted at Fig. 4, there are four possible infinite paths in this transition system, but none of them visits both ■ and ●. It does not give us a pseudo-Nash equilibrium and in fact there is none in this game.

4 Application to Finite Games

In this section, we apply the previous generic procedure to finite concurrent games. We consider four classes of finite concurrent games: $\mathfrak{C}^{\mathrm{nd}}$ is the whole class of finite concurrent non-deterministic games, $\mathfrak{C}^{\mathrm{d}}$ is the restriction to deterministic games, $\mathfrak{TB}^{\mathrm{nd}}$ is the restriction to turn-based games, and $\mathfrak{TB}^{\mathrm{d}}$ is the intersection of $\mathfrak{C}^{\mathrm{d}}$ and $\mathfrak{TB}^{\mathrm{nd}}$.

We also consider subclasses where the number of players is bounded *a priori*, and thus is not taken into account in the complexity. Our results can be summarized as follows (in grey are previously known results [7, Corollary 1][3]):

	$\mathfrak{C}^{\mathrm{nd}}$, $\mathfrak{C}^{\mathrm{d}}$, $\mathfrak{TB}^{\mathrm{nd}}$		$\mathfrak{TB}^{\mathrm{d}}$	
	bounded	general	bounded	general
Existence	P-c.	NP-c.	True	True
Verification	P-c.	NP-c.	P-c.	NP-c.
Constr. Ex.	P-c.	NP-c.	P-c.	NP-c.

These results all follow from the following Proposition:

[3] The results in [17] concern parity objectives, and do not encompass reachability objectives.

Proposition 13. *1. The following problems are* P*-hard with bounded number of players and* NP*-hard in the general case:*

(a) checking that a payoff ν *corresponds to a Nash equilibrium in* $\mathfrak{TB}^d$*;*

(b) deciding the existence of a pseudo-Nash equilibrium in $\mathfrak{TB}^{nd}$*;*

(c) deciding the existence of a Nash equilibrium in $\mathfrak{C}^d$*.*

2. Solving the constrained-existence problem in $\mathfrak{C}^{nd}$ *is in* P *for a bounded number of players, and in* NP *in the general case.*

Proof (Sketch of proof). P- and NP-hardness results are obtained by straightforward encodings of the CIRCUIT-VALUE and 3SAT problems, respectively.

The NP algorithm for the constrained existence problem is obtained by first guessing the payoff function, and then checking that Theorem 11 holds. This is achieved by guessing a sequence of states in $\mathcal{S}(P)$, and checking that it is indeed a path in $\mathcal{S}(P)$ and that it visits the correct sets in Ω_{Agt}. A naive implementation of this procedure runs in exponential time (because computing $\mathcal{S}(P)$ may require the computation of intermediate sets $\mathsf{Rep}(P \cap P')$ for many subsets P' of Agt, which may result in up to $2^{|P|}$ computation steps), but using non-determinism, we can select polynomially many intermediate repellor sets that must be computed. The procedure thus runs in non-deterministic polynomial time.

In the case where the number of agents is bounded, the naive approach above is already polynomial, and the number of payoff functions is also polynomial. We can then enumerate all payoff functions, build the transition system $\mathcal{S}(P)$ for each of them, and check the existence of a "witness" path in this transition system. □

Remark 14. In the case of turn-based games, the set of suspects is always either empty, or a singleton, or the whole set of players. As a consequence, the naive implementation of the procedure above will not result in computing $2^{|P|}$ repellor sets, but only $|P|$. The global algorithm still runs in NP, because finding a path in $\mathcal{S}(P)$ with several reachability constraints is NP-complete.

5 Application to Timed Games

5.1 Definitions of Timed Games

A *valuation* over a finite set of clocks X is an application $v\colon X \to \mathbb{R}_+$. If v is a valuation and $t \in \mathbb{R}_+$, then $v + t$ is the valuation that assigns to each $x \in X$ the value $v(x) + t$. If v is a valuation and $Y \subseteq X$, then $[Y \leftarrow 0]v$ is the valuation that assigns 0 to each $y \in Y$ and $v(x)$ to each $x \in X \setminus Y$. A *clock constraint* over X is a formula built on the grammar: $\mathfrak{C}(X) \ni g \ ::= \ x \sim c \ \mid \ g \wedge g$, where x ranges over X, $\sim \in \{<, \leq, =, \geq, >\}$, and c is an integer. The semantics of clock constraints over valuations is natural, and we omit it.

We now define the notion of timed games that we will use in this paper. Our definition follows that of [8].

Definition 15. *A* timed game *is a* 7*-tuple* $\mathcal{G} = \langle \mathsf{Loc}, X, \mathsf{Inv}, \mathsf{Trans}, \mathsf{Agt}, \mathsf{Owner}, \preccurlyeq\rangle$ *where:*

- Loc *is a finite set of locations;*
- X *is a finite set of* clocks*;*
- $\mathsf{Inv}\colon \mathsf{Loc} \to \mathfrak{C}(X)$ *assigns an* invariant *to each location;*
- $\mathsf{Trans} \subseteq \mathsf{Loc} \times \mathfrak{C}(clocks) \times 2^X \times \mathsf{Loc}$ *is the set of transitions;*
- Agt *is a finite set of agents (or players);*
- $\mathsf{Owner}\colon \mathsf{Trans} \to \mathsf{Agt}$ *assigns an agent to each transition;*
- $\preccurlyeq\colon \mathsf{Agt} \to 2^{(\mathsf{States}\times\mathbb{R}_+)^\omega \times (\mathsf{States}\times\mathbb{R}_+)^\omega}$ *defines, for each player, a quasi-ordering on the runs of* $\mathcal{G}$*, called* preference relation.

As in the previous sections, we restrict here to the case where $\preccurlyeq$ is given in terms of reachability objectives $(\varOmega_A)_{A\in\mathsf{Agt}}$, with $\varOmega_A \subseteq \mathsf{Loc}$ for each $A \in \mathsf{Agt}$.

A timed game is played as follows: a state of the game is a pair (ℓ, v) where ℓ is a location and v is a clock valuation, provided that $v \models \mathsf{Inv}(\ell)$. From each state (starting from an initial state $s_0 = (\ell, \mathbf{0})$, where $\mathbf{0}$ maps each clock to zero and is assumed to satisfy $\mathsf{Inv}(\ell)$), each player A chooses a nonnegative real number d and a transition δ, with the intended meaning that she wants to delay for d time units and then fire transition δ. There are several (natural) restrictions on these choices:

- spending d time units in ℓ must be allowed[4] *i.e.*, $v + d \models \mathsf{Inv}(\ell)$;
- $\delta = (\ell, g, z, \ell')$ belongs to player A, *i.e.*, $\mathsf{Owner}(\delta) = A$;
- the transition is firable after d time units (*i.e.*, $v + d \models g$), and the invariant is satisfied when entering ℓ' (*i.e.*, $[z \leftarrow 0](v + d) \models \mathsf{Inv}(\ell')$).

If (and only if) there is no such possible choice for some player A (*e.g.* if no transition from ℓ belongs to A), then she chooses a special move, denoted by $\perp$.

Given the set of choices m_{Agt} of all the players, with $m_A \in (\mathbb{R}_+ \times \mathsf{Trans}) \cup \{\perp\}$, a player B such that $d_B = \min\{d_A \mid A \in \mathsf{Agt} \text{ and } m_A = (d_A, \delta_A)\}$ is selected (non-deterministically), and the corresponding transition δ_B is applied, leading to a new state $(\ell', [z \leftarrow 0](v + d))$.

This semantics can naturally be expressed in terms of an infinite-state non-deterministic concurrent game. Timed games inherit the notions of history, play, path, strategy, profile, outcome and (pseudo-)Nash equilibrium *via* this correspondence.

In the sequel, we consider only *non-blocking* timed games, *i.e.*, timed games in which, for any reachable state (ℓ, v), at least one player has an allowed action:

$$\prod_{A\in\mathsf{Agt}} \mathsf{Mov}((\ell, v), A) \neq \{(\perp)_{A\in\mathsf{Agt}}\}.$$

[4] Formally, this should be written $v + d' \models \mathsf{Inv}(\ell)$ for all $0 \leq d' \leq d$, but this is equivalent to having only $v \models \mathsf{Inv}(\ell)$ and $v + d \models \mathsf{Inv}(\ell)$ since invariants are convex.

5.2 Computing Pseudo-nash Equilibria in Timed Games

Let $\mathcal{G} = \langle \mathsf{Loc}, X, \mathsf{Inv}, \mathsf{Trans}, \mathsf{Agt}, \mathsf{Owner}, \preccurlyeq\rangle$ be a timed game, where $\preccurlyeq$ is given in terms of reachability objectives $(\Omega_A)_{A\in\mathsf{Agt}}$. In this section, we explain how pseudo-Nash equilibria can be computed in such reachability timed games, using Theorem 11. This relies on the classical notion of *regions* [1], which we assume the reader is familiar with.

We define the region game $\mathcal{R} = \langle \mathsf{States}_\mathcal{R}, \mathsf{Edg}_\mathcal{R}, \mathsf{Agt}, \mathsf{Act}_\mathcal{R}, \mathsf{Mov}_\mathcal{R}, \mathsf{Tab}_\mathcal{R}, \preccurlyeq_\mathcal{R}\rangle$ as follows:

- $\mathsf{States}_\mathcal{R} = \{(\ell, r) \in \mathsf{Loc} \times \mathfrak{R} \mid r \models \mathsf{Inv}(\ell)\}$, where $\mathfrak{R}$ is the set of clock regions;
- $\mathsf{Edg}_\mathcal{R}$ is the set of transitions of the region automaton underlying $\mathcal{G}$;
- $\mathsf{Act}_\mathcal{R} = \{(r, p, \delta) \mid r \in \mathfrak{R},\ p \in \{1; 2; 3\} \text{ and } \delta \in \mathsf{Trans}\} \cup \{\bot\}$;
- $\mathsf{Mov}_\mathcal{R} \colon \mathsf{States}_\mathcal{R} \times \mathsf{Agt} \to 2^{\mathsf{Act}_\mathcal{R}} \setminus \{\varnothing\}$ is such that

$$\begin{aligned}\mathsf{Mov}_\mathcal{R}((\ell, r), A) = \{(r', p, \delta) \mid{} & r' \in \mathsf{Succ}(r),\ r' \models \mathsf{Inv}(\ell),\\ & p \in \{1; 2; 3\} \text{ if } r' \text{ is time-elapsing, else } p = 1,\\ & \delta = (\ell, g, z, \ell') \in \mathsf{Trans} \text{ is such that } r' \models g\\ & \qquad \text{and } [z \leftarrow 0]r' \models \mathsf{Inv}(\ell')\text{and } \mathsf{Owner}(\delta) = A\}\end{aligned}$$

  if it is non-empty, and $\mathsf{Mov}_\mathcal{R}((\ell, r), A) = \{\bot\}$ otherwise. Roughly, the index p allows the players to say if they want to play first, second or later if their region is selected.
- $\mathsf{Tab}_\mathcal{R} \colon \mathsf{States}_\mathcal{R} \times {\mathsf{Act}_\mathcal{R}}^{\mathsf{Agt}} \to 2^{\mathsf{Edg}_\mathcal{R}} \setminus \{\varnothing\}$ is such that for every $(\ell, r) \in \mathsf{States}_\mathcal{R}$ and every $m_\mathsf{Agt} \in \prod_{A\in\mathsf{Agt}} \mathsf{Mov}_\mathcal{R}((\ell, r), A)$, if we write r' for[5] $\min\{r_A \mid m_A = (r_A, p_A, \delta_A)\}$ and p' for $\min\{p_A \mid m_A = (r', p_A, \delta_A)\}$,

$$\begin{aligned}\mathsf{Tab}_\mathcal{R}((\ell, r), m_\mathsf{Agt}) = \{&((\ell, r), (\ell_B, [z_B \leftarrow 0]r_B)) \mid\\ & m_B = (r_B, p_B, \delta_B) \text{ with } r_B = r',\ p_B = p' \text{ and } \delta_B = (\ell, g_B, z_B, \ell_B)\}.\end{aligned}$$

- The preference relation $\preccurlyeq_\mathcal{R}$ is defined in terms of reachability objectives for each player, where the set of objectives $(\Omega'_i)_{A_i\in\mathsf{Agt}}$ $(\Omega'_A)_{A\in\mathsf{Agt}}$ is defined, for each $A \in \mathsf{Agt}$, as $\Omega'_A = \{(\ell, r) \mid \ell \in \Omega_A, r \in \mathfrak{R}\}$.

Proposition 16. *Let $\mathcal{G}$ be a timed game, and $\mathcal{R}$ its associated region game. Then there is a pseudo-Nash equilibrium in $\mathcal{G}$ from $(s, \mathbf{0})$ iff there is a pseudo-Nash equilibria in $\mathcal{R}$ from $(s, [\mathbf{0}])$, where $[\mathbf{0}]$ is the region associated to $\mathbf{0}$. Furthermore, this equivalence is constructive.*

Proof (Sketch of proof). The proof is in three steps: we first define a kind of generic *simulation* relation between games, which gives information on their respective repellor sets and transition systems:

[5] This is well-defined for two reasons: first, not all m_i's may be $\bot$, since we consider non-blocking games; second, the set of regions appearing in a move from (ℓ, r) only contains successors of r, and is then totally ordered.

Lemma 17. *Consider two games $\mathcal{G}$ and $\mathcal{G}'$ involving the same set of agents, with preference relations defined in terms of reachability conditions $(\Omega_A)_{A\in\mathsf{Agt}}$ and $(\Omega'_A)_{A\in\mathsf{Agt}}$, respectively. Assume that there exists a binary relation $\lhd$ between states of $\mathcal{G}$ and states of $\mathcal{G}'$ such that, if $s \lhd s'$, then:*

1. *if $s' \in \Omega'_A$ then $s \in \Omega_A$ for any $A \in \mathsf{Agt}$;*
2. *for all move m_{Agt} in $\mathcal{G}$, there exists a move m'_{Agt} in $\mathcal{G}'$ such that:*
   - *for any t' in $\mathcal{G}'$, there is $t \lhd t'$ in $\mathcal{G}$ s.t. $\mathsf{Susp}((s',t'),m'_{\mathsf{Agt}}) \subseteq \mathsf{Susp}((s,t),m_{\mathsf{Agt}})$;*
   - *for any (s,t) in $\mathsf{Tab}(s,m_{\mathsf{Agt}})$, there is a (s',t') in $\mathsf{Tab}(s',m'_{\mathsf{Agt}})$ s.t. $t \lhd t'$.*

Then for any $P \subseteq \mathsf{Agt}$ and for any s and s' such that $s \lhd s'$, it holds:

1. *if s is in $\mathsf{Rep}_{\mathcal{G}}(P)$, then s' is in $\mathsf{Rep}_{\mathcal{G}'}(P)$;*
2. *for any $(s,t) \in \mathsf{Edg}_{\mathsf{Rep}}$, there exists (s',t') in $\mathsf{Edg}'_{\mathsf{Rep}}$ s.t. $t \lhd t'$, where $\mathsf{Edg}_{\mathsf{Rep}}$ and $\mathsf{Edg}'_{\mathsf{Rep}}$ are the set of edges in the transition systems $\mathcal{S}_{\mathcal{G}}(P)$ and $\mathcal{S}_{\mathcal{G}'}(P)$, respectively.*

It remains to show that a timed game and its associated region game simulate one another in the sense of Lemma 17, which entail that they have the same sets of repellors. This is achieved by defining two functions λ, mapping moves in $\mathcal{G}$ to *equivalent* moves in $\mathcal{R}$, and μ, mapping moves in $\mathcal{R}$ to *equivalent* moves in $\mathcal{G}$, in such a way that $v \lhd r$ iff r is the region containing v. Theorem 11 concludes the proof. □

Because the region game $\mathcal{R}$ has size exponential in the size of $\mathcal{G}$, we get:

Theorem 18. *The constrained existence problem (and thus the existence- and verification problems) in timed game can be solved in EXPTIME.*

Remark 19. Given a pseudo-Nash equilibrium $(\alpha_{\mathsf{Agt}}, \pi)$ in the region game, we can obtain one in the timed game for the same payoff vector. Assume that $(\alpha_{\mathsf{Agt}}, \pi)$ is a pseudo-Nash equilibrium in $\mathcal{R}$. Given a history h in $\mathcal{G}$ and its projection $\mathsf{proj}(h)$ in $\mathcal{R}$, if $(\alpha_A(\mathsf{proj}(h)))_{A\in\mathsf{Agt}} = (r_A, p_A, \delta_A)_{A\in\mathsf{Agt}}$ is a secure move in $\mathcal{R}$, then so is $(\mu_A)_{A\in\mathsf{Agt}} = \mu(\mathsf{last}(h), (\alpha_A(\mathsf{proj}(h)))_{A\in\mathsf{Agt}})$, where μ is the function used in the proof of Proposition 16 to simulate moves from $\mathcal{R}$ in $\mathcal{G}$. Moreover, there exists a play $\pi' \in \mathsf{Out}((\ell, v), (\mu_A)_{A\in\mathsf{Agt}})$ such that $\mathsf{proj}(\pi') = \pi$, therefore the payoff function for these two plays is the same. Hence $((\mu_A)_{A\in\mathsf{Agt}}), \pi')$ is a pseudo-Nash equilibrium in the timed game.

Our algorithm is optimal, as we prove EXPTIME-hardness of our problems:

Proposition 20. *The constrained-existence and verification problems for deterministic turn-based timed games with at least two clocks and two players is EXPTIME-hard. The existence problem is EXPTIME-hard for concurrent timed games (with at least two clocks and two players).*

This is proved by encoding countdown games [11]. The second part of the Proposition requires the use of a timed game with no equilibria; an example of such a game is depicted on Fig. 5.

Remark 21. Since deterministic turn-based timed games yield deterministic turn-based region games, they always admit a Nash equilibrium.

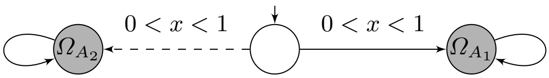


Fig. 5. A timed game with no equilibria (the solid transition belongs to the first player, the dashed one to the second one)

6 Conclusion

In this paper we have described a procedure to compute qualitative pseudo-Nash equilibria in multi-player concurrent (possibly non-deterministic) games with reachability objectives. The development of this procedure has required technical tools as the repellor sets, which can be seen as an alternative to the classical attractor sets for computing equilibria in games. We have applied this procedure to finite concurrent games and to timed games, yielding concrete algorithms to compute equilibria in those games. We have furthermore proved that those algorithms have optimal complexities.

Multiple extensions of this work are rather natural:

- First we would like to apply the generic procedure to other classes of systems, for instance to pushdown games [20]. Note that we are not aware of any result on the computation of equilibria in pushdown games.
- Then our procedure only applies to reachability objectives for every player. It would be interesting to adapt it to other ω-regular winning objectives. This is *a priori* non-trivial as this will require developing new tools (the repellor sets are dedicated to reachability objectives).
- We have applied our procedure to concurrent games as defined *e.g.* in [2], where the transition table of the game is given in extensive form (for each tuple of possible actions, there is an entry in a table). In [12], a more compact way of representing concurrent game is proposed, which assumes a symbolic representation of the transition table. It would be interesting to study how this does impact on the complexity of the computation of Nash equilibria. In particular the argument for having an NP algorithm (Proposition 13) does not hold anymore.
- Finally other notions of equilibria (subgame-perfect equilibria, secure equilibria, *etc*) could be investigated, and extensions of concurrent games with probabilities could also be investigated.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. TCS 126(2), 183–235 (1994)
2. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM 49, 672–713 (2002)
3. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: Proc. IFAC Symp. System Structure and Control, pp. 469–474. Elsevier, Amsterdam (1998)

4. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 121–125. Springer, Heidelberg (2007)
5. Bouyer, P., Brenguier, R., Markey, N.: Nash equilibria for reachability objectives in multi-player timed games. Research report LSV-10-12, Lab. Spécification & Vérification, ENS Cachan, France (June 2010)
6. Chatterjee, K., Henzinger, T.A., Jurdziński, M.: Games with secure equilibria. In: LICS 2006, pp. 160–169. IEEE Comp. Soc. Press, Los Alamitos (2006)
7. Chatterjee, K., Majumdar, R., Jurdziński, M.: On Nash equilibria in stochastic games. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 26–40. Springer, Heidelberg (2004)
8. de Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: The element of surprise in timed games. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 142–156. Springer, Heidelberg (2003)
9. Félegyházi, M., Hubaux, J.-P., Buttyán, L.: Nash equilibria of packet forwarding strategies in wireless ad hoc networks. IEEE Trans. Mobile Computing 5(5), 463–476 (2006)
10. Henzinger, T.A.: Games in system design and verification. In: TARK 2005, pp. 1–4. Nat. Univ., Singapore (2005)
11. Jurdziński, M., Laroussinie, F., Sproston, J.: Model checking probabilistic timed automata with one or two clocks. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 170–184. Springer, Heidelberg (2007)
12. Laroussinie, F., Markey, N., Oreiby, G.: On the expressiveness and complexity of ATL. LMCS 4(2:7) (2008)
13. Nash, J.F.: Equilibrium points in n-person games. Proc. National Academy of Sciences of the USA 36(1), 48–49 (1950)
14. Paul, S., Simon, S.: Nash equilibrium in generalised Muller games. In: FSTTCS 2009. LIPIcs, vol. 4, pp. 335–346. LZI (2009)
15. Thomas, W.: Infinite games and verification (extended abstract of a tutoral). In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 58–64. Springer, Heidelberg (2002)
16. Ummels, M.: Rational behaviour and strategy construction in infinite multiplayer games. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 212–223. Springer, Heidelberg (2006)
17. Ummels, M.: The complexity of Nash equilibria in infinite multiplayer games. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 20–34. Springer, Heidelberg (2008)
18. Ummels, M., Wojtczak, D.: The complexity of Nash equilibria in simple stochastic multiplayer games. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 297–308. Springer, Heidelberg (2009)
19. Ummels, M., Wojtczak, D.: Decision problems for Nash equilibria in stochastic games. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 515–530. Springer, Heidelberg (2009)
20. Walukiewicz, I.: Pushdown processes: Games and model checking. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 234–263. Springer, Heidelberg (1996)

Stochastic Real-Time Games with Qualitative Timed Automata Objectives*

Tomáš Brázdil, Jan Krčál, Jan Křetínský**, Antonín Kučera, and Vojtěch Řehák

Faculty of Informatics, Masaryk University, Brno, Czech Republic
{brazdil,krcal,kucera,rehak}@fi.muni.cz, jan.kretinsky@in.tum.de

Abstract. We consider two-player stochastic games over real-time probabilistic processes where the winning objective is specified by a timed automaton. The goal of player □ is to play in such a way that the play (a timed word) is accepted by the timed automaton with probability one. Player ◇ aims at the opposite. We prove that whenever player □ has a winning strategy, then she also has a strategy that can be specified by a timed automaton. The strategy automaton reads the history of a play, and the decisions taken by the strategy depend only on the region of the resulting configuration. We also give an exponential-time algorithm which computes a winning timed automaton strategy if it exists.

1 Introduction

In this paper, we study *stochastic real-time games (SRTGs)* which are obtained as a natural game-theoretic extension of *generalized semi-Markov processes (GSMP)* [13,20,21] or *real-time probabilistic processes (RTP)* [2]. Intuitively, all of these formalisms model systems which react to certain *events*, such as message receipts, subsystem failures, timeouts, etc. A common characteristic of all events is that they are *delayed* (it takes some time before an initiated event actually occurs) and *concurrent* (there can be several previously initiated events that are currently awaited). For example, if two messages e and e' are sent, it takes some (random) time before they arrive, and one can specify, or approximate, the *densities* f_e, $f_{e'}$ of their arrival times. When e arrives (say, after 20 time units), the system reacts to this event by changing its state, and awaits e' in a new state. The arrival time of e' in the new state is measured from zero again, and its density $f_{e'|20}$ is obtained from $f_{e'}$ by incorporating the condition that e' is delayed for at least 20 time units. That is, $f_{e'|20}(x) = f_e(x + 20)/\int_{20}^{\infty} f_e(y)\,dy$. Note that if the delays of all events are exponentially distributed, then $f_e = f_{e|b}$ for every $b \in \mathbb{R}_{\geq 0}$, and thus we obtain continuous-time Markov chains (see, e.g., [17]) and continuous-time stochastic games [10,18] as restricted forms of RTPs and SRTGs, respectively.

Intuitively, a SRTG is a finite graph (see Fig. 1) with three types of nodes—*states* (drawn as large circles), *controls*, where each control can be either *internal* or *adversarial* (drawn as boxes and diamonds, respectively), and *actions* (drawn as small filled

* The authors are supported by the Alexander von Humboldt Foundation (T. Brázdil), the Institute for Theoretical Computer Science, project No. 1M0545 (J. Krčál), Brno Municipality (J. Křetínský), and the Czech Science Foundation, grants No. P202/10/1469 (A. Kučera) and No. 201/08/P459 (V. Řehák).

** On leave at TU München, Boltzmannstr. 3, Garching, Germany.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 207–221, 2010.

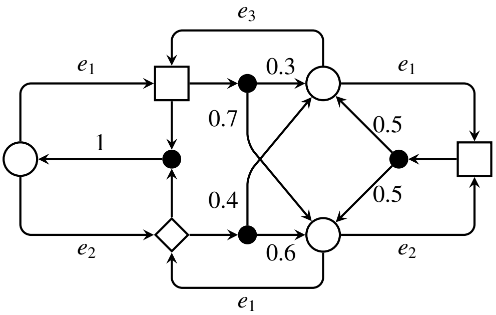


Fig. 1. An example of a stochastic real-time game

circles). In each state s, there is a finite subset $E(s)$ of events *scheduled* in s (the events scheduled in s are those which are "awaited" in a given state; the other events are disabled. Each state s can react to every event of $E(s)$ by entering a designated control c, where player □ or player ◇ chooses some of the available actions. Each action is associated with a fixed probability distribution over states. In general, both players can use randomized strategies, which means that they do not necessarily select just a single action but a probability distribution over the available actions, which is multiplied with the distributions associated to actions. Then, the next state is chosen randomly according to the constructed probability distribution, and the play goes on. Whenever a new state s' is entered from a previous state s along a play, each event scheduled in s' is assigned a new *delay* which is chosen randomly according to the corresponding (conditional) density. The state s' then "reacts" to the event with the least delay (under the assumptions adopted in this paper, the probability of assigning the same delay to different events is zero).

Our contribution. In this work we consider SRTGs with *deterministic timed automata (DTA)* objectives. Intuitively, a timed automaton "observes" a play of a given SRTG and checks that certain timing constraints are satisfied. A simple example of a property that can be encoded by a DTA is "whenever a new request is generated, it is either serviced within the next 10 time units, or the system eventually enters a safe state". In this case, we want to setup the internal controls so that the above property holds for almost all plays, no matter what decisions are taken in adversarial controls. Hence, the aim of player □ is to maximize the probability that a play is accepted by a given timed automaton, while player ◇ aims at the opposite. By applying the result of [14], we obtain that SRTGs with DTA objectives have a *value*, i.e., $\sup_\sigma \inf_\pi P^{\sigma,\pi} = \inf_\pi \sup_\sigma P^{\sigma,\pi}$, where σ and π range over all strategies of player □ and player ◇, and $P^{\sigma,\pi}$ is the probability of all plays satisfying a given DTA objective. This immediately raises the question whether the players have *optimal* strategies which guarantee the equilibrium value against every strategy of the opponent. We show that the answer is *negative*. Then, we concentrate on the *qualitative* variant of the problem, which is perhaps most interesting from the practical point of view. An *almost-sure winning* strategy for player □ is a strategy such that for every strategy of player ◇, the probability of all plays satisfying a given DTA objective is equal to one. The main result of this paper is the following: We show that if player □ has *some* almost-sure winning strategy, then she also has a *DTA* almost-sure winning strategy, which can be encoded by a deterministic timed automaton $\mathcal{A}$ constructible in exponential time. The automaton $\mathcal{A}$ reads the history of a play, and the decision taken by the corresponding DTA strategy depends only on the region of the resulting configuration entered by $\mathcal{A}$.

Our constructions and proofs are combinations of standard techniques (used for timed automata and finite-state games) and some new non-trivial observations that are specific for the considered model of SRTGs. We also adapt some ideas presented in [2] (in particular, we use the concept of δ-separation).

Related work. Continuous-time (semi)Markov chains are a classical and deeply studied model with a mature mathematical theory (see, e.g., [17,19]). Continuous-time Markov decision processes (CTMDPs) [7,5,16] combine probabilistic and non-deterministic choice, but all events are required to be exponentially distributed. Two player games over continuous-time Markov chains were considered only recently [10,18]. Timed automata [3] were originally introduced as a non-stochastic model with time. Probabilistic semantics of timed automata was proposed in [4,6], and a more general model of stochastic games over timed automata was considered in [9]. In this paper we build mainly on the previous work about GSMPs [13,20,21] and RTPs [2,1] and interpret timed automata as a model-independent specification language which can express important properties of timed systems. This view is adopted also in [12] where continuous-time Markov chains are checked against timed-automata specifications.

Let us note that our technical treatment of events is somewhat different from the one used for GSMPs and RTPs. Intuitively, in GSMPs (and RTPs), each event is assigned its delay only when it is newly scheduled, and this delay is just updated when moving from state to state (by subtracting the elapsed time) until the event happens or it is disabled. For example, if two messages e and e' are sent, both of them are assigned randomly chosen delays d_e and $d_{e'}$. The smaller of the two delays (say d_e) triggers a transition to the next state, where the delay of $d_{e'}$ is updated by subtracting d_e. Since the current delays of all events are explicitly recorded in the state-space of GSMPs and RTPs, this formalism cannot be directly extended to perfect-information games (the players would "see" the delays assigned to events, i.e., they would know what is going to happen in the future). In our model of SRTGs, we always assign a new random delay to all events that are scheduled in a given control state, but we adjust the corresponding densities (from a "probabilistic" point of view, this approach is equivalent to the one used for GSMPs and RTPs).

Due to space constraints, most of the proofs are omitted and can be found in a full version of this paper [11].

2 Definitions

In this paper, the sets of all positive integers, non-negative integers, real numbers, positive real numbers, and non-negative real numbers are denoted by $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{R}$, $\mathbb{R}_{>0}$, and $\mathbb{R}_{\geq 0}$, respectively.

Let A be a finite or countably infinite set. A *probability distribution* on A is a function $f : A \to \mathbb{R}_{\geq 0}$ such that $\sum_{a \in A} f(a) = 1$. We say that f is *rational* if $f(a)$ is rational for every $a \in A$. The set of all distributions on A is denoted by $\mathcal{D}(A)$. A *σ-field* over a set Ω is a set $\mathcal{F} \subseteq 2^{\Omega}$ that includes Ω and is closed under complement and countable union. A *measurable space* is a pair $(\Omega, \mathcal{F})$ where Ω is a set called *sample space* and $\mathcal{F}$ is a σ-field over Ω whose elements are called *measurable sets*. A *probability measure* over a measurable space $(\Omega, \mathcal{F})$ 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 $\mathcal{P}(\Omega) = 1$. A *probability space* is a triple $(\Omega, \mathcal{F}, \mathcal{P})$, where $(\Omega, \mathcal{F})$ is a measurable space and $\mathcal{P}$ is a probability measure over $(\Omega, \mathcal{F})$. We say that a property $A \subseteq \Omega$ holds *for almost all* elements of a measurable set Y if $\mathcal{P}(Y) > 0$, $A \cap Y \in \mathcal{F}$, and $\mathcal{P}(A \mid Y) = 1$.

Let us note that all of the integrals used in this paper should be understood as Lebesgue integrals, although we use Riemann-like notation.

2.1 Stochastic Real-Time Games

Let $\mathcal{E}$ be a finite set of *events*, which are independent of each other. To every $e \in \mathcal{E}$ we associate its *lower bound* $\ell_e \in \mathbb{N}_0$, *upper bound* $u_e \in \mathbb{N} \cup \{\infty\}$, and a *density function* $f_e : \mathbb{R} \to \mathbb{R}$ which is positive on (ℓ_e, u_e) such that $\int_{\ell_e}^{u_e} f_e(x)\,dx = 1$. Further, for every $b \in \mathbb{R}_{\geq 0}$ we also define the *conditional density function* $f_{e|b} : \mathbb{R} \to \mathbb{R}$ as follows:

$$f_{e|b}(x) = \frac{f_e(x+b)}{\left[\int_b^{u_e} f_e(y)\,dy\right]_{\neq 0}}$$

Here $[\cdot]_{\neq 0} : \mathbb{R} \to \mathbb{R}$ is a function which for a given x returns either x or 1 depending on whether $x \neq 0$ or not, respectively. The function f_e defines the density of delaying the event e, i.e., for every time $t \in \mathbb{R}_{\geq 0}$, the probability of delaying e for at most t is equal to $\int_0^t f_e(x)\,dx$. Note that the integral $\int_0^t f_{e|b}(x)\,dx$ is equal to the conditional probability of delaying e for at most $b + t$ under the condition that e is delayed for at least b. Since all events are mutually independent, for every subset $E' \subseteq E$ we have that the conditional probability of delaying all events in E' for at least $b + t$ under the condition that all events in E' are delayed for at least b is equal to $\prod_{e \in E'} \int_t^{\infty} f_{e|b}(x)\,dx$.

Definition 1. *A* stochastic real-time game (SRTG) *is a tuple* $\mathcal{G} = (S, E, C_{\Box}, C_{\Diamond}, Act, F, A, \mu_0)$ *where* S *is a finite set of* states, $E : S \to 2^{\mathcal{E}}$ *assigns to each* $s \in S$ *the set of events scheduled to occur in* s, $C_{\Box}$ *and* $C_{\Diamond}$ *are finite disjoint sets of* controls *of player* $\Box$ *and player* $\Diamond$, $Act \subseteq \mathcal{D}(S)$ *is a finite set of* actions, F *is a* flow function *which to every pair* (s, e), *where* $s \in S$ *and* $e \in E(s)$, *assigns a control of* $C_{\Box} \cup C_{\Diamond}$, $A : C_{\Box} \cup C_{\Diamond} \to 2^{Act}$ *assigns to each control c a non-empty finite set of actions enabled at c, and* $\mu_0 \in \mathcal{D}(S)$ *is an* initial *distribution.*

A *stamp* is an element (s, t, e) of $S \times \mathbb{R}_{>0} \times \mathcal{E}$ where $e \in E(s)$. A (computational) *history* of $\mathcal{G}$ is a finite sequence $\mathfrak{h} = (s_0, t_0, e_0), \ldots, (s_n, t_n, e_n)$ of stamps. Intuitively, t_i is the time spent in s_i while waiting for some of the events scheduled in s_i, and e_i is the event that triggered a transition to the next state s_{i+1}. A *strategy* of player $\odot$, where $\odot \in \{\Box, \Diamond\}$, is a measurable function which to every history $(s_0, t_0, e_0), \ldots, (s_n, t_n, e_n)$ such that $F(s_n, e_n) = c \in C_{\odot}$ assigns a probability distribution over the set $A(c)$ of actions that are enabled at c. The set of all strategies of player $\Box$ and player $\Diamond$ are denoted by Σ and Π, respectively.

Let $(\sigma, \pi) \in \Sigma \times \Pi$. The corresponding *play* of $\mathcal{G}$ is initiated in some $s_0 \in S$ (with probability $\mu_0(s_0)$). Then, each event $e \in E(s_0)$ is assigned a randomly chosen delay $d_e^0 \in \mathbb{R}_{>0}$ according to the density f_e (note that $f_e = f_{e|0}$). Let $t_0 = \min\{d_e^0 \mid e \in E(s_0)\}$ be the minimal delay of all events scheduled in s_0, and let $\mathit{trigger}_0$ be the set of all $e \in E(s_0)$ such that $d_e^0 = t_0$. The event e_0 which "triggers" a transition to the next state is the least element of $\mathit{trigger}_0$ w.r.t. some fixed linear ordering $\preceq$ (note that the probability of assigning the same delay to different events is zero, and hence the choice of $\preceq$ is irrelevant; we need this ordering just to make our semantics well defined). The event e_0 determines a control $c = F(s_0, e_0)$, where the responsible player makes a decision according to her strategy τ, i.e., selects a distribution $\tau(\mathfrak{h})$ over $A(c)$ where $\mathfrak{h} = (s_0, t_0, e_0)$ is the current history. Hence, the next state s_1 is chosen with probability $\sum_{\mu \in A(c)} \tau(\mathfrak{h})(\mu) \cdot \mu(s_1)$. In s_1, we assign a randomly chosen delay d_e^1 to every $e \in E(s_1)$ according to the conditional density $f_{e|b}$, where b is determined as follows: If e was scheduled in the previous state s_0 and $e \neq e_0$, then $b = t_0$; otherwise $b = 0$. The event e_1 is the least event (w.r.t. $\preceq$) with the minimal delay $t_1 = \min\{d_e^1 \mid e \in E(s_1)\}$. The next state s_2 is chosen randomly by combining the strategy of the respective player with the corresponding actions. In general, after entering a state s_i, every $e \in E(s_i)$ is assigned a randomly chosen delay d_e^i according to the conditional density $f_{e|b}$ where b is the total waiting time for e accumulated in the history of the play.

To formalize the intuition given above, we define a suitable probability space $(\mathit{Play}, \mathcal{F}, \mathcal{P}_{\mathfrak{h}}^{\sigma,\pi})$ over the set Play of all infinite sequences of stamps, where $\mathfrak{h}$ is a history of steps "performed previously" (the technical convenience of $\mathfrak{h}$ becomes apparent later in Section 3; the definition given below is perhaps easier to understand in the special case when $\mathfrak{h}$ is empty). For the rest of this section, we fix a history $\mathfrak{h} = (s_0, t_0, e_0), \ldots, (s_n, t_n, e_n)$ where $n \in \mathbb{N}_0 \cup \{-1\}$. If $n = -1$, then $\mathfrak{h}$ is empty. A *template* is a finite sequence of the form $B = (s_{n+1}, I_{n+1}, e_{n+1}), \ldots, (s_{n+m}, I_{n+m}, e_{n+m})$ such that $m \geq 1$, $e_i \in E(s_i)$, and I_i is an interval in $\mathbb{R}_{>0}$ for every $n+1 \leq i \leq n+m$. Each such B determines the corresponding *cylinder* $\mathit{Play}(B) \subseteq \mathit{Play}$ consisting of all sequences of the form $(s_{n+1}, t_{n+1}, e_{n+1}), \ldots, (s_{n+m}, t_{n+m}, e_{n+m}), \ldots$ where $t_i \in I_i$ for all $n+1 \leq i \leq n+m$. The σ-field $\mathcal{F}$ is the Borel σ-field generated by all cylinders. For each cylinder $\mathit{Play}(B)$, the probability $\mathcal{P}_{\mathfrak{h}}^{\sigma,\pi}(\mathit{Play}(B))$ is defined in the way described below. Then, $\mathcal{P}_{\mathfrak{h}}^{\sigma,\pi}$ is extended to $\mathcal{F}$ (in the unique way) by applying the extension theorem (see, e.g., [8]).

It remains to show how to define the probability $\mathcal{P}_{\mathfrak{h}}^{\sigma,\pi}(\mathit{Play}(B))$ of a given cylinder $\mathit{Play}(B)$, where $B = (s_{n+1}, I_{n+1}, e_{n+1}), \ldots, (s_{n+m}, I_{n+m}, e_{n+m})$. We put $\mathcal{P}_{\mathfrak{h}}^{\sigma,\pi}(\mathit{Play}(B)) = T_{n+1}$, where the expression T_i is defined inductively for all $n+1 \leq i \leq n+m+1$ as follows:

$$T_i = \begin{cases} \int_{I_i} \mathit{State}_i \cdot \mathit{Win}_i \cdot T_{i+1}\, dt_i & \text{if } n+1 \leq i \leq n+m; \\ 1 & \text{if } i = n+m+1. \end{cases}$$

Observe that T_{n+1} is an expression with m nested integrals. Further, note that when constructing T_{i+1}, we already have $t_0, \ldots, t_i$ at our disposal (each t_i is either fixed in $\mathfrak{h}$, or it is a variable used in some of the preceding integrals).

The subterm State_i corresponds to the probability that s_i is chosen as the next state, assuming that the current history is $(s_0, t_0, e_0), \ldots, (s_{i-1}, t_{i-1}, e_{i-1})$. Hence, we define

- $State_{n+1} = \mu_0(s_{n+1})$ if $\mathfrak{h}$ is empty, otherwise $State_{n+1} = \sum_{\mu \in A(c)} \tau(\mathfrak{h})(\mu) \cdot \mu(s_{n+1})$, where $c = F(s_n, e_n)$, and τ is either σ or π, depending on whether $c \in C_\Box$ or $c \in C_\Diamond$, respectively.
- $State_i = \sum_{\mu \in A(c)} \tau(\mathfrak{h}')(\mu) \cdot \mu(s_i)$, where $n{+}1 < i \leq n{+}m$, $c = F(s_{i-1}, e_{i-1})$, $\mathfrak{h}' = (s_0, t_0, e_0), \ldots, (s_{i-1}, t_{i-1}, e_{i-1})$, and τ is either σ or π, depending on whether $c \in C_\Box$ or $c \in C_\Diamond$, respectively.

The most complicated part is the definition of Win_i which intuitively corresponds to the probability that the event e_i "wins" the competition among the events scheduled in s_i.

In order to define Win_i, we have to overcome a technical obstacle that the events scheduled in s_i might have been scheduled also in the preceding states. For each $e \in E(s_i)$, let $K(e,i)$ be the minimal index such that $0 \leq K(e,i) \leq i$ and for all $K(e,i) \leq j < i$ we have that $e \in E(s_j)$ and $e \neq e_j$. We put $b(e,i) = t_{K(e,i)} + \cdots + t_{i-1}$. Intuitively, $b(e,i)$ is the total waiting time for e accumulated in the history of the play. Note that if $K(e,i) = i$, then the defining sum of $b(e,i)$ is empty and hence equal to zero. We put

$$Win_i \quad = \quad f_{e_i|b(e_i,i)}(t_i) \quad \cdot \prod_{\substack{e \in E(s_i) \\ e \neq e_i}} \int_{t_i}^{\infty} f_{e|b(e,i)}(x)\, dx.$$

2.2 Deterministic Timed Automata

Let $\mathcal{X}$ be a finite set of *clocks*. A *valuation* is a function $\nu : \mathcal{X} \to \mathbb{R}_{\geq 0}$. For every valuation ν and every subset $X \subseteq \mathcal{X}$ of clocks, we use $\nu[X := \mathbf{0}]$ to denote the unique valuation such that $\nu[X := \mathbf{0}](x) = 0$ for all $x \in X$, and $\nu[X := \mathbf{0}](x) = \nu(x)$ for all $x \in \mathcal{X} \setminus X$. Further, for every valuation ν and every $\delta \in \mathbb{R}_{\geq 0}$, the symbol $\nu + \delta$ denotes the unique valuation such that $(\nu + \delta)(x) = \nu(x) + \delta$ for all $x \in \mathcal{X}$.

A *clock constraint* (or *guard*) is a finite conjunction of basic constraints of the form $x \bowtie c$, where $x \in \mathcal{X}$, $\bowtie \in \{<, \leq, >, \geq\}$, and $c \in \mathbb{N}_0$. For every valuation ν and every clock constraint g we have that ν either does or does not satisfy g, written $\nu \models g$ or $\nu \not\models g$, respectively (the satisfaction relation is defined in the expected way). Sometimes we slightly abuse our notation and identify a guard g with the set of all valuations that satisfy g (for example, we write $g \cap g'$). The set of all guards over $\mathcal{X}$ is denoted by $\mathcal{B}(\mathcal{X})$.

Definition 2. *A* deterministic timed automaton (DTA) *is a tuple* $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \longrightarrow, q_0, T)$*, where* Q *is a nonempty finite set of* locations*,* Σ *is a finite* alphabet*,* $\mathcal{X}$ *is a finite set of* clocks*,* $q_0 \in Q$ *is an* initial location*,* $T \subseteq Q$ *is a set of* target locations*, and* $\longrightarrow \subseteq Q \times \Sigma \times \mathcal{B}(\mathcal{X}) \times 2^{\mathcal{X}} \times Q$ *is an* edge relation *such that for all* $q \in Q$ *and* $a \in \Sigma$ *we have the following:*

1. *the guards are deterministic, i.e., for all edges of the form* (q, a, g_1, X_1, q_1) *and* (q, a, g_2, X_2, q_2) *such that* $g_1 \cap g_2 \neq \emptyset$ *we have that* $g_1 = g_2$*,* $X_1 = X_2$*, and* $q_1 = q_2$*;*
2. *the guards are total, i.e., for all* $q \in Q$*,* $a \in \Sigma$*, and every valuation* ν *there is an edge* (q, a, g, X, q') *such that* $\nu \models g$*.*

A *configuration* of $\mathcal{A}$ is a pair (q, ν), where $q \in Q$ and ν is a valuation. An *infinite timed word* is an infinite sequence $w = c_0 c_1 c_2 \ldots$ where each c_i is either a letter of Σ or a positive real number denoting a time stamp (note that letters and time stamps are not required to alternate in w). The *run* of $\mathcal{A}$ on w is the unique infinite sequence $(q_0, \nu_0)\ c_0\ (q_1, \nu_1)\ c_1\ \ldots$ such that q_0 is the initial location of $\mathcal{A}$, $\nu_0 = \mathbf{0}$, and for each $i \in \mathbb{N}_0$ we have that

- if c_i is a time stamp $t \in \mathbb{R}_{\geq 0}$, then $q_{i+1} = q_i$ and $\nu_{i+1} = \nu_i + t$,
- if c_i is an input letter $a \in \Sigma$, then there is a unique edge (q_i, a, g, X, q) such that $\nu_i \models g$, and we require that $q_{i+1} = q$ and $\nu_{i+1} = \nu_i[X := \mathbf{0}]$.

We say that w is *accepted* by $\mathcal{A}$ if the run of $\mathcal{A}$ on w visits a configuration (q, ν) where $q \in T$. Without restrictions, we may assume that each $q \in T$ is *absorbing*, i.e., all of the outgoing edges of q lead back to q.

In this paper, we use DTA for two different purposes. Firstly, DTA are used as a generic *specification language* for properties of timed systems. In this case, a given DTA is constructed so that it accepts the set of all "correct" runs (timed words) of a given timed system. Formally, for a fixed SRTG $\mathcal{G}$ with a set of states S, a finite set Ap of atomic propositions and a labeling $L : S \to 2^{Ap}$, every play $\varrho = (s_0, t_0, e_0), (s_1, t_1, e_1), \ldots$ of $\mathcal{G}$ determines a unique infinite timed word $Ap(\varrho) = L(s_0)\, t_0\, L(s_1)\, t_1 \ldots$. A DTA $\mathcal{A}$ with alphabet 2^{Ap} then either accepts $Ap(\varrho)$ or not. Intuitively, the automaton $\mathcal{A}$ encodes some desirable property of plays, and the aim of player $\Box$ and player $\Diamond$ is to maximize and minimize the probability of all plays accepted by $\mathcal{A}$, respectively. We denote $Play(\mathcal{A}) \subseteq Play$ the set of all plays ϱ such that $Ap(\varrho)$ is accepted by $\mathcal{A}$. Note that the DTA does not read any information about the events that occurred. However, one can easily encode the information about the last event into the subsequent state by considering copies s_e of each state s for every event e.

Secondly, we use DTA to encode strategies in stochastic real-time games. Here, the constructed DTA "observes" the history of a play, and the decisions taken by the corresponding strategy depend only on the resulting configuration (q, ν). Actually, we require that the decision depends only on the *region* of (q, ν) (see [3] or Section 3.1), which makes DTA strategies finitely representable. Formally, every history $\mathfrak{h} = (s_0, t_0, e_0) \cdots (s_n, t_n, e_n)$ of $\mathcal{G}$ can be seen as a (finite) timed word $s_0, t_0, e_0, \cdots, s_n, t_n, e_n$, where the states and events are seen as letters, and the delays are seen as time stamps. We define DTA strategies as follows.

Definition 3. *A* DTA strategy *is a strategy τ such that there is a DTA $\mathcal{A}$ with alphabet $S \cup \mathcal{E}$ satisfying the following: for every history $\mathfrak{h}$ we have that $\tau(\mathfrak{h})$ is a rational distribution which depends only on the region of (q, ν), where (q, ν) is the configuration entered by $\mathcal{A}$ after reading the word $\mathfrak{h}$.*

3 Results

For the rest of the paper, we fix an SRTG $\mathcal{G} = (S, E, C_\Box, C_\Diamond, Act, F, A, \mu_0)$, a finite set Ap of atomic propositions, a labeling $L : S \to 2^{Ap}$, and a DTA $\mathcal{A} = (Q, 2^{Ap}, \mathcal{X}, \longrightarrow, q_0, T)$.

As observed in [14], the determinacy result for Blackwell games [15] implies determinacy of a large class of stochastic games. This abstract class includes the games studied in this paper, and thus we obtain the following:

Proposition 1. *Let $\mathfrak{h}$ be a history of $\mathcal{G}$. Then*

$$\sup_{\sigma \in \Sigma} \inf_{\pi \in \Pi} \mathcal{P}^{\sigma,\pi}_{\mathfrak{h}}(Play(\mathcal{A})) \quad = \quad \inf_{\pi \in \Pi} \sup_{\sigma \in \Sigma} \mathcal{P}^{\sigma,\pi}_{\mathfrak{h}}(Play(\mathcal{A}))$$

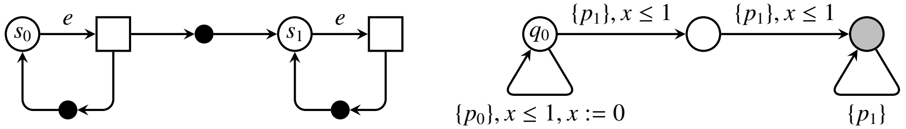


Fig. 2. Player □ does not have an optimal strategy

The value *of $\mathcal{G}$ (with respect to $\mathfrak{h}$), denoted by $val_{\mathfrak{h}}$, is defined by the above equality.*

The existence of $val_{\mathfrak{h}}$ implies the existence of ε-optimal strategies for both players. However, note that player □ does not necessarily have an *optimal* strategy which would achieve the outcome $val_{\mathfrak{h}}$ or better against every strategy of player ◇, even if $val_{\mathfrak{h}} = 1$ and $C_\diamond = \emptyset$. A simple counterexample is given in Fig. 2. Here f_e is the uniform density on $(0, 1)$ (i.e., $f_e(x) = 1$ for all $x \in (0, 1)$), $Ap = \{p_0, p_1\}$, $L(s_0) = p_0$, $L(s_1) = p_1$, and the only target location is gray. All of the "missing" edges in the depicted DTA (which are needed to satisfy the requirement that the guards are total) lead to a "garbage" location. The initial distribution μ_0 assigns 1 to s_0. Now observe that $val_{\mathfrak{h}} = 1$ (where $\mathfrak{h}$ is the empty history), because for every $\varepsilon > 0$, player □ can "wait" in s_0 until e is fired so that its delay is smaller than ε (this eventually happens with probability 1), and then she moves to s_1. The probability that e is assigned a delay at most $1 - \varepsilon$ in s_1 is $1 - \varepsilon$, and hence the constructed DFA accepts a play with probability $1 - \varepsilon$. However, player □ has no optimal strategy.

In this paper we consider the existence and effective constructability of *almost-sure* winning strategies for player □. Formally, a strategy $\sigma \in \Sigma$ is almost-sure winning for a history $\mathfrak{h}$ if for every strategy $\pi \in \Pi$ we have that $\mathcal{P}^{\sigma,\pi}_{\mathfrak{h}}(Play(\mathcal{A})) = 1$. We show the following:

Theorem 1. *Let $\mathfrak{h}$ be a history. If player □ has (some) almost-sure winning strategy for $\mathfrak{h}$, then she also has a DTA almost-sure winning strategy for $\mathfrak{h}$. The existence of a DTA almost-sure winning strategy for $\mathfrak{h}$ is decidable in exponential time, and if it exists, it can be constructed in exponential time.*

A proof of Theorem 1 is not immediate and requires several steps. First, in Section 3.1 we construct a *product game* $\mathcal{G}_{\mathcal{A}}$ of $\mathcal{G}$ and $\mathcal{A}$ and show that $\mathcal{G}_{\mathcal{A}}$ can be examined instead of $\mathcal{G}$ and $\mathcal{A}$. The existence of a DTA almost-sure winning strategy in $\mathcal{G}_{\mathcal{A}}$ is analyzed in Section 3.2. Finally, in Section 3.3 we present an algorithm which computes a DTA almost-sure winning strategy if it exists.

3.1 The Product Game

Intuitively, the product game of $\mathcal{G}$ and $\mathcal{A}$, denoted by $\mathcal{G}_{\mathcal{A}}$, is constructed by simulating the execution of $\mathcal{A}$ on-the-fly in $\mathcal{G}$. Waiting times for events and clock valuations are represented explicitly in the states of $\mathcal{G}_{\mathcal{A}}$, and hence the state-space of $\mathcal{G}_{\mathcal{A}}$ is uncountable. Still, $\mathcal{G}_{\mathcal{A}}$ is in many aspects similar to $\mathcal{G}$, and therefore we use a suggestive notation compatible with the one used for $\mathcal{G}$. To distinguish among the notions related to $\mathcal{G}$ and $\mathcal{G}_{\mathcal{A}}$, we consistently use the "p-" prefix. Hence, $\mathcal{G}$ has stamps, states, histories, etc., while $\mathcal{G}_{\mathcal{A}}$ has p-stamps, p-states, p-histories, etc.

Let $n = |\mathcal{E}| + |\mathcal{X}|$. The clock values of $\mathcal{A}$ and the delays of currently scheduled events are represented by a *p-vector* $\xi \in \mathbb{R}^n_{\geq 0}$. The set of *p-states* is $S \times Q \times \mathbb{R}^n_{\geq 0}$, and the sets of *p-controls* of player □ and player ◇ are $C_\Box \times Q \times \mathbb{R}^n_{\geq 0}$ and $C_\Diamond \times Q \times \mathbb{R}^n_{\geq 0}$, respectively.

The dynamics of $\mathcal{G}_\mathcal{A}$ is determined as follows. First, we define a *p-flow* function $F_\mathcal{A}$, which to a given p-stamp (s, q, ξ, t, e) assigns the p-control (c, q', ξ'), where $c = F(s, e)$, and q', ξ' are determined as follows. Let $(q, L(s), g, X, q')$ be the unique edge of $\mathcal{A}$ such that the guard g is satisfied by the clock valuation stored in $\xi + t$. We put $\xi' = (\xi +_s t)[(e \cup X) := \mathbf{0}]$. The operator "$+_s t$" adds t to all clocks stored in ξ and to all events scheduled in s, and $(e \cup X) := \mathbf{0}$ resets all clocks of X to zero and assigns zero delay to e. Second, we define the set of p-actions. For every p-control (c, q, ξ) and an action $a \in A(c)$, there is a corresponding p-action which to a given p-state (s', q, ξ'), where $\xi' = \xi[(\mathcal{E} \setminus E(s')) := \mathbf{0}]$, assigns the probability $a(s')$.

A *p-stamp* is an element (s, q, ξ, t, e) of $S \times Q \times \mathbb{R}^n_{\geq 0} \times \mathbb{R}_{>0} \times \mathcal{E}$. Now we define *p-histories* and *p-plays* as sequences of p-stamps. In the game $\mathcal{G}$ we allowed arbitrary sequences of stamps, whereas in the product game we need the automaton part of the product to be consistent with the game part. We say that a p-stamp $x_1 = (s_1, q_1, \xi_1, t_1, e_1)$ is consistent with a p-stamp $x_0 = (s_0, q_0, \xi_0, t_0, e_0)$ if the image of x_0 under the p-flow function is a p-control (c, q_1, ξ') such that $\xi_1 = \xi'[A := \mathbf{0}]$ where A is the set of actions not enabled in s_1.

A *p-history* is a finite sequence of p-stamps $\mathfrak{p} = x_0 \ldots x_n$ such that x_i is consistent with x_{i+1} for all $0 \leq i < n$. A *p-play* is an infinite sequence of p-stamps $x_0 x_1 \ldots$ where each finite prefix $x_0 \ldots x_i$ is a p-history. Each p-history $\mathfrak{p} = (s_0, q_0, \xi_0, t_0, e_0), \ldots, (s_n, q_n, \xi_n, t_n, e_n)$ can be mapped to a unique history $H(\mathfrak{p}) = (s_0, t_0, e_0), \ldots, (s_n, t_n, e_n)$. Note that H is in fact a bijection, because each history induces a unique finite execution of the DTA $\mathcal{A}$ and the consistency condition reflects this unique execution. By the *last p-control* of a p-history $\mathfrak{p}$ we denote the image of the last p-stamp of $\mathfrak{p}$ under the p-flow function.

Region relation. Although the state-space of $\mathcal{G}_\mathcal{A}$ is uncountable, we can define a variant of *region relation* over p-histories which has a finite index, and then work with finitely many *regions*.

For a given $x \in \mathbb{R}_{\geq 0}$, we use *frac*$(x)$ to denote the fractional part of x, and *int*(x) to denote the integral part of x. For $x, y \in \mathbb{R}_{\geq 0}$, we say that x and y *agree on integral part* if *int*(x) = *int*(y) and neither or both x, y are integers. A *relevant bound* of a clock x is the largest constant c that appears in all guards. A *relevant bound* of an event e is u_e if $u_e < \infty$, and ℓ_e otherwise. We say that an element $a \in \mathcal{E} \cup \mathcal{X}$ is *relevant* for ξ if $\xi(a) \leq r$ where r is the relevant bound of a. Finally, we put $\xi_1 \approx \xi_2$ if

- for all relevant $a \in \mathcal{E} \cup \mathcal{X}$ we have that $\xi_1(a)$ and $\xi_2(a)$ agree on integral parts;
- for all relevant $a, b \in \mathcal{E} \cup \mathcal{X}$ we have that *frac*$(\xi_1(a)) \leq$ *frac*$(\xi_1(b))$ if and only if *frac*$(\xi_2(a)) \leq$ *frac*$(\xi_2(b))$.

The equivalence classes of $\approx$ are called *time areas*. Now we can define the promised *region relation* $\sim$ on p-histories. Let $\mathfrak{p}_1$ and $\mathfrak{p}_2$ be p-histories such that (c_1, q_1, ξ_1) is the last p-control of $\mathfrak{p}_1$ and (c_2, q_2, ξ_2) is the last p-control of $\mathfrak{p}_2$. We put $\mathfrak{p}_1 \sim \mathfrak{p}_2$ iff $c_1 = c_2$, $q_1 = q_2$ and $\xi_1 \approx \xi_2$. Note that $\sim$ is an equivalence with a finite index. The equivalence classes of $\sim$ are called *regions*. A *target region* is a region that contains such p-histories whose last p-controls have a target location in the second component. The sets of all regions and target regions are denoted by $\mathcal{R}$ and $\mathcal{R}_T$, respectively.

Remark 1. Let us note that the region construction described above can also be applied to configurations of timed automata, where it coincides with the standard region construction of [3].

Strategies in the product game. Note that every pair of strategies $(\sigma, \pi) \in \Sigma \times \Pi$ defined for the original game $\mathcal{G}$ can also be applied in the constructed product game $\mathcal{G}_{\mathcal{A}}$ (we just ignore the extra components of p-stamps). By re-using the construction of Section 2.1, for every p-history $\mathfrak{p}$ and every pair of strategies $(\sigma, \pi) \in \Sigma \times \Pi$, we define a probability measure $\mathcal{P}^{\sigma,\pi}_{\mathfrak{p}}$ on the Borel σ-field $\mathcal{F}$ over the p-plays in $\mathcal{G}_{\mathcal{A}}$ (the details are given in [11]).

For every $\mathcal{S} \subseteq \mathcal{R}$, let $Reach(\mathcal{S})$ be the set of all p-plays that visit a region of $\mathcal{S}$ (i.e., some prefix of the p-play belongs to some $r \in \mathcal{S}$). We say that a strategy $\sigma \in \Sigma$ is *almost-sure winning* in $\mathcal{G}_{\mathcal{A}}$ for a p-history $\mathfrak{p}$ if for every $\pi \in \Pi$ we have that $\mathcal{P}^{\sigma,\pi}_{\mathfrak{p}}(Reach(\mathcal{R}_T)) = 1$. The relationship between almost-sure winning strategies in $\mathcal{G}$ and $\mathcal{G}_{\mathcal{A}}$ is formulated in the next proposition.

Proposition 2. *Let $\sigma \in \Sigma$ and $\mathfrak{p}$ be a p-history. Then σ is almost-sure winning for $\mathfrak{p}$ in $\mathcal{G}_{\mathcal{A}}$ iff σ is almost-sure winning for $H(\mathfrak{p})$ in $\mathcal{G}$.*

Another observation about strategies in $\mathcal{G}_{\mathcal{A}}$ which is heavily used in the next sections concerns strategies that are *constant on regions*. Formally, a strategy $\tau \in \Sigma \cup \Pi$ is constant on regions if for all p-histories $\mathfrak{p}_1$ and $\mathfrak{p}_2$ such that $\mathfrak{p}_1 \sim \mathfrak{p}_2$ we have that $\tau(\mathfrak{p}_1) = \tau(\mathfrak{p}_2)$.

Proposition 3. *Every strategy $\tau \in \Sigma \cup \Pi$ which is constant on regions is a DTA strategy.*

Proof (Sketch). We transform τ into a DTA $A_{\mathcal{G}_{\mathcal{A}}}$ whose regions are in one-to-one correspondence with the regions of $\mathcal{G}_{\mathcal{A}}$. The automaton $A_{\mathcal{G}_{\mathcal{A}}}$ reads a sequence of stamps of $\mathcal{G}$ and simulates the behavior of $\mathcal{G}_{\mathcal{A}}$. It has a special clock for every clock of $\mathcal{A}$ and every event of $\mathcal{E}$, and uses its locations to store also the current state of the game. The details are given in [11]. □

Note that due to Proposition 3, every strategy constant on regions can be effectively transformed into a DTA strategy.

3.2 Almost-Sure Winning Strategies

In this section, we outline a proof of the following theorem:

Theorem 2. *Let $\mathfrak{p}$ be a p-history. If there is a strategy $\sigma \in \Sigma$ which is almost-sure winning in $\mathcal{G}_{\mathcal{A}}$ for $\mathfrak{p}$, then there is a DTA strategy $\sigma^* \in \Sigma$ which is almost-sure winning for $\mathfrak{p}$.*

Note that due to Proposition 3, it suffices to show that there is an almost-sure winning strategy in $\mathcal{G}_{\mathcal{A}}$ for $\mathfrak{p}$ which is constant on regions.

Observe that if $\sigma \in \Sigma$ is an almost-sure winning strategy in $\mathcal{G}_{\mathcal{A}}$ for $\mathfrak{p}$, then for every $\pi \in \Pi$ the plays of $\mathcal{G}_{\mathcal{A}}$ may visit only regions from which it is still possible to visit a target region. Hence, a good candidate for an almost-sure winning DTA strategy in $\mathcal{G}_{\mathcal{A}}$ for $\mathfrak{p}$ is a strategy which never leaves this set of "safe" regions. This motivates the following definition (in the rest of this section we often write $\mathfrak{p} \in \mathcal{S}$, where $\mathfrak{p}$ is a p-history and $\mathcal{S}$ a set of regions, to indicate that $\mathfrak{p} \in \bigcup_{r \in \mathcal{S}} r$).

Definition 4. *A DTA strategy $\sigma \in \Sigma$ is a* candidate on a set of regions $\mathcal{S} \subseteq \mathcal{R}$ *if for every $\pi \in \Pi$ and every p-history $\mathfrak{p} \in \mathcal{S}$ we have that $\mathcal{P}^{\sigma,\pi}_{\mathfrak{p}}(Reach(\mathcal{R} \setminus \mathcal{S})) = 0$ and $\mathcal{P}^{\sigma,\pi}_{\mathfrak{p}}(Reach(\mathcal{R}_T)) > 0$.*

In the following, we prove Propositions 4 and 5 that together imply Theorem 2.

Proposition 4. *Let σ be an almost-sure winning strategy in $\mathcal{G}_{\mathcal{A}}$ for a p-history $\mathfrak{p}_0$. Then there is a set $\mathcal{S} \subseteq \mathcal{R}$ and a DTA strategy σ^* such that $\mathfrak{p}_0 \in \mathcal{S}$ and σ^* is a candidate on $\mathcal{S}$.*

Proof (Sketch). We define $\mathcal{S}$ as the set of all regions reached with positive probability in an arbitrary play where player □ uses the strategy σ and player ◇ uses some $\pi \in \Pi$. For every action a, let $p\text{-}hist_a$ be the set of all p-histories where σ assigns a positive probability to a. For every region $r \in \mathcal{S}$, we denote by A_r the set of all $a \in Act$ for which there is $\pi \in \Pi$ such that $\mathcal{P}^{\sigma,\pi}_{\mathfrak{p}_0}(p\text{-}hist_a \cap r) > 0$.

- Firstly, we show that every DTA strategy σ' that selects only the actions of A_r in every $r \in \mathcal{S}$ has to satisfy $\mathcal{P}^{\sigma',\pi}_{\mathfrak{p}}(Reach(\mathcal{R} \setminus \mathcal{S})) = 0$ for all $\pi \in \Pi$ and $\mathfrak{p} \in \mathcal{S}$. To see this, realize that when we use only the actions of A_r, we do not visit (with positive probability) any other regions than we did with σ. Hence, we stay in $\mathcal{S}$ almost surely.
- Secondly, we prove that from every p-history in $\mathcal{S}$, we can reach a target region with positive probability. We proceed in several steps.
  - Let us fix a region $r \in \mathcal{S}$. Realize that then there is a p-history $\mathfrak{p} \in r$ for which σ is almost-sure winning (since σ is almost-sure winning and for every $r \in \mathcal{S}$ there is $\pi \in \Pi$ such that r is visited with positive probability, there must be a p-history $\mathfrak{p} \in r$ for which σ is almost-sure winning). In particular, $\mathcal{P}^{\sigma,\pi}_{\mathfrak{p}}(Reach(\mathcal{R}_T)) > 0$ for every $\pi \in \Pi$. We show how to transform σ into a DTA strategy σ' such that $\mathcal{P}^{\sigma',\pi}_{\mathfrak{p}}(Reach(\mathcal{R}_T)) > 0$.

    Let us first consider one-player games, i.e., the situation when $C_\diamond = \emptyset$. Then there must be a sequence of regions $r_0, \ldots, r_n$ visited on the way from $\mathfrak{p}$ to a target, selecting some actions $a_0, \ldots, a_{n-1}$. We fix these actions for the respective regions (if some region is visited several times, we fix the last action taken) and thus obtain the desired DTA strategy σ'.

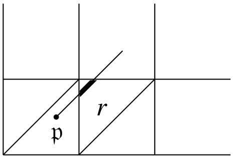
    In the general case of two-player games, we have to consider a tree of regions and actions instead of a single sequence, because every possible behaviour of the opponent in the first n steps has to be taken into account.
  - Then we prove that for *every* $\mathfrak{p}' \in r$ we have that $\mathcal{P}^{\sigma',\pi}_{\mathfrak{p}'}(Reach(\mathcal{R}_T)) > 0$ for every $\pi \in \Pi$. For the p-histories $\mathfrak{p}, \mathfrak{p}' \in r$, consider the probability that taking an action a results in reaching a given region in one step. These probabilities are either both positive or both zero. This one-step qualitative equivalence is then extended to arbitrary many steps. Hence, $\mathcal{P}^{\sigma',\pi}_{\mathfrak{p}'}(Reach(\mathcal{R}_T)) > 0$.

- Let us now drop the fixed region r. We need to "stitch" the DTA strategies described above for each region into one DTA strategy σ^*. We construct σ^* as follows. In the first step, we take an arbitrary region reachable with positive probability (e.g., the initial one containing $\mathfrak{p}_0$) and fix the decisions in the regions $r_0, \ldots, r_n$ (where $r_n \in \mathcal{R}_T$) discussed above. Let us denote this set of regions by F_1. In the second step, we take an arbitrary region $v \in \mathcal{S} \setminus F_1$. Again, we take a sequence of regions $r'_0, \ldots, r'_{n'}$ ending in $\mathcal{R}_T \cup F_1$. We fix the actions in these regions accordingly and get a set F_2. We repeat this step until $F_k = \mathcal{S}$. In the other regions, σ^* is defined arbitrarily requiring only it is constant on each region. □

Proposition 5. *If a DTA strategy σ^* is a candidate on a set of regions $\mathcal{S} \subseteq \mathcal{R}$, then for every $\mathfrak{p} \in \mathcal{S}$ and every $\pi \in \Pi$ we have that $\mathcal{P}_{\mathfrak{p}}^{\sigma^*,\pi}(Reach(\mathcal{R}_T)) = 1$.*

Note that we are guaranteed that for every p-history in every region in $\mathcal{S}$, the probability of reaching a target is positive. However, it can be arbitrarily small. Therefore, even if we pass through these regions infinitely often and never leave them, it is not clear that we eventually reach a target almost surely. This would be guaranteed if the probabilities were bounded from below by a positive constant.

Remark 2. If we considered the restricted case of one-player games with bounded intervals and exponentially distributed unbounded events, we can already easily prove that σ^* is winning using [3] as follows. Fixing σ^* resolves all non-determinism and yields a system of the type considered by [3]. Since we are guaranteed the positive probability of reaching the target, we may apply Lemma 3 of [3]. However, in the setting of two-player games, we cannot use this argument directly and some (non-trivial) changes are required.



Intuitively, the reason why the probabilities of reaching a target are generally not bounded from below is that when the fractional parts of the clocks are too close, the probability of reaching a given region may approach zero. The figure on the left shows the region graph of a system with two clocks and a single state. There is also a single event, which is positive on (0, 1) and its associated clock is not depicted. Now observe that if $\mathfrak{p}$ comes closer and closer to the diagonal, the probability that the (only) event happens in the region r is smaller and smaller.

Nevertheless, we can bound the probabilities if we restrict ourselves to a smaller set of positions. We define *δ-separated* parts of regions, where the differences of p-clocks are at least δ (and hence we are at least δ-away from the boundary of the region) or zero due to a synchronization of the clocks of the original automaton. Being away from the boundary by a fixed δ then guarantees that we reach the next region with a probability bounded from below.

Definition 5. *Let $\delta > 0$. We say that a set $D \subseteq \mathbb{R}_{\geq 0}$ is δ-separated if for every $x, y \in D$ either $frac(x) = frac(y)$ or $|frac(x) - frac(y)| > \delta$. Further, we say that a p-history with the last p-control (s, q, ξ) is δ-separated if the set $\{0\} \cup \{\xi(a) \mid a \in \mathcal{E} \cup \mathcal{X}, a \text{ is relevant for } \xi\}$ is δ-separated.*

Now we prove that the probabilities of reaching a target region are bounded from below if we start in a δ-separated p-history.

Proposition 6. *Let σ^* be a DTA strategy candidate on a set of regions $\mathcal{S}$. For every $\delta > 0$ there is $\varepsilon > 0$ such that for every δ-separated p-history $\mathfrak{p} \in \mathcal{S}$ and every strategy π we have that $\mathcal{P}_{\mathfrak{p}}^{\sigma^*,\pi}(Reach(\mathcal{R}_T)) > \varepsilon$.*

Proof (Sketch). We prove that for every $\delta > 0$ there is $\varepsilon > 0$ such that starting in a δ-separated p-history, the probability of reaching a target in at most $|\mathcal{R}|$ steps is greater than ε. For this we use the observation that after performing one step from a δ-separated p-history, we end up (with a probability bounded from below) in a δ'-separated p-history. This can be generalized to an arbitrary (but fixed) number of steps. Now it suffices to observe that for every $\pi \in \Pi$ and a δ-separated p-history $\mathfrak{p}$ there is a sequence of regions $r_1, \ldots, r_k$ with $k \leq |\mathcal{R}|$, such that $\mathfrak{p} \in r_1$, $r_k \in \mathcal{R}_T$, and the probability of reaching r_{i+1} from r_i in one step using σ^* and π is positive. □

Nevertheless, there is a non-zero probability of falling out of safely separated parts of regions. To finish the proof of Proposition 5, we need to know that we pass through δ-separated p-histories infinitely often almost surely (since the probability of reaching a target from δ-separated p-histories is bounded from below by Proposition 6, a target is eventually visited with probability one). For this, it suffices to prove that we eventually return to a δ-separated part almost surely. Hence, the following proposition makes our proof complete.

Proposition 7. *There is $\delta > 0$ such that for every DTA strategy $\sigma \in \Sigma$ and every $\pi \in \Pi$, a δ-separated p-history is reached almost surely from every p-history $\mathfrak{p}$.*

Proof (Sketch). We prove that there are $n \in \mathbb{N}$, $\delta > 0$, and $\varepsilon > 0$ such that for every p-history $\mathfrak{p}$ and every $\pi \in \Pi$, the probability of reaching a δ-separated p-history in n steps is greater than ε. Then, we just iterate the argument. □

3.3 The Algorithm

In this section, we show that the existence of a DTA almost-sure winning strategy is decidable in exponential time, and we also show how to compute such a strategy if it exists. Due to Proposition 2, this problem can be equivalently considered in the setting of the product game $\mathcal{G}_{\mathcal{A}}$. Due to Proposition 3, an almost-sure winning DTA strategy can be constructed as a strategy that is constant on every region of $\mathcal{G}_{\mathcal{A}}$. We show that this problem can be further reduced to the problem of computing wining strategies in a *finite* stochastic game $\mathcal{G}^{\mathcal{A}}$ with reachability objectives induced by the product game $\mathcal{G}_{\mathcal{A}}$. Note that the game $\mathcal{G}^{\mathcal{A}}$ can be solved by standard methods (e.g., by computing the attractor of a target set). First, we define the game $\mathcal{G}^{\mathcal{A}}$ and show how to compute it. The complexity discussion follows.

The product $\mathcal{G}_{\mathcal{A}}$ induces a game $\mathcal{G}^{\mathcal{A}}$ whose vertices are the regions of $\mathcal{G}_{\mathcal{A}}$ as follows. Player $\odot$, where $\odot \in \{\Box, \Diamond\}$, plays in regions $(c, q, [\xi]_{\approx})$ [1] where $c \in C_{\odot}$. In a region

[1] Note that a region is a set of p-histories such that their last p-controls share the same control c, location q, and equivalence class $[\xi]_{\approx}$. Hence, we can represent a region by a triple $(c, q, [\xi]_{\approx})$.

$r = (c, q, [\xi]_{\approx})$, she chooses an arbitrary action $a \in A(c)$ and this action a leads to a stochastic vertex $(r, a) = ((c, q, [\xi]_{\approx}), a)$. From this stochastic vertex there are transitions to all regions $r' = (c', q', [\xi']_{\approx})$, such that r' is reachable from all $\mathfrak{p} \in r$ in one step using action a with some positive probability in the product $\mathcal{G}_{\mathcal{A}}$. One of these probabilistic transitions is taken at random according to the uniform distribution. From the next region the play continues in the same manner. Player □ tries to reach the set $\mathcal{R}_T$ of target regions (which is the same as in the product game) and player ◇ tries to avoid it. We say that a strategy σ of player □ is almost-sure winning for a vertex v if she reaches $\mathcal{R}_T$ almost surely when starting from v and playing according to σ.

At first glance, it might seem surprising that we set all probability distributions in $\mathcal{G}^{\mathcal{A}}$ as uniform. Note that in different parts of a region r, the probabilities of moving to r' are different. However, as noted in the sketch of proof of Proposition 4, they are all positive or all zero. Since we are interested only in *qualitative* reachability, this is sufficient for our purposes.

Moreover, note that since we are interested in non-zero probability behaviour, there are no transitions to regions which are reachable only with zero probability (such as when an event occurs at an integral time).

We now prove that the reduction is correct. Observe that a strategy for the product game $\mathcal{G}_{\mathcal{A}}$ which is constant on regions induces a unique positional strategy for the game $\mathcal{G}^{\mathcal{A}}$, and vice versa. Slightly abusing the notation, we consider these strategies to be strategies in both games.

Proposition 8. *Let $\mathcal{G}$ be a game and $\mathcal{A}$ a deterministic timed automaton. For every p-history $\mathfrak{p}$ in a region r, we have that*

- *a positional strategy σ is almost-sure winning for r in $\mathcal{G}^{\mathcal{A}}$ iff it is almost-sure winning for $\mathfrak{p}$ in $\mathcal{G}_{\mathcal{A}}$,*
- *player □ has an almost-sure winning strategy for r in $\mathcal{G}^{\mathcal{A}}$ iff player □ has an almost-sure winning strategy for $\mathfrak{p}$ in $\mathcal{G}_{\mathcal{A}}$.*

The algorithm constructs the regions of the product $\mathcal{G}_{\mathcal{A}}$ and the induced game graph of the game $\mathcal{G}^{\mathcal{A}}$ (see [11]). Since there are exponentially many regions (w.r.t. the number of clocks and events), the size of $\mathcal{G}^{\mathcal{A}}$ is exponential in the size of $\mathcal{G}$ and $\mathcal{A}$. As we already noted, two-player stochastic games with qualitative reachability objectives are easily solvable in polynomial time, and thus we obtain the following:

Theorem 3. *Let $\mathfrak{h}$ be a history. The problem whether player □ has a (DTA) almost-sure winning strategy for $\mathfrak{h}$ is solvable in time exponential in $|\mathcal{G}|$ and $|\mathcal{A}|$, and polynomial in $|\mathfrak{h}|$. A DTA almost-sure winning strategy is computable in exponential time if it exists.*

4 Conclusions and Future Work

An interesting question is whether the positive results presented in this paper can be extended to more general classes of objectives that can be encoded, e.g., by deterministic timed automata with ω-regular acceptance conditions. Another open problem are algorithmic properties of ε-optimal strategies in stochastic real-time games.

References

1. Alur, R., Courcoubetis, C., Dill, D.L.: Model-checking for probabilistic real-time systems. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) ICALP 1991. LNCS, vol. 510, pp. 115–136. Springer, Heidelberg (1991)
2. Alur, R., Courcoubetis, C., Dill, D.L.: Verifying automata specifications of probabilistic real-time systems. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 28–44. Springer, Heidelberg (1992)
3. Alur, R., Dill, D.: A theory of timed automata. TCS 126(2), 183–235 (1994)
4. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T., Größer, M.: Almost-sure model checking of infinite paths in one-clock timed automata. In: Proceedings of LICS 2008, pp. 217–226. IEEE, Los Alamitos (2008)
5. Baier, C., Hermanns, H., Katoen, J.-P., Haverkort, B.R.: Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. TCS 345, 2–26 (2005)
6. Bertrand, N., Bouyer, P., Brihaye, T., Markey, N.: Quantitative model-checking of one-clock timed automata under probabilistic semantics. In: Proceedings of 5th Int. Conf. on Quantitative Evaluation of Systems (QEST 2008), pp. 55–64. IEEE, Los Alamitos (2008)
7. Bertsekas, D.P.: Dynamic Programming and Optimal Control. Athena Scientific, Belmont (2007)
8. Billingsley, P.: Probability and Measure. Wiley, Chichester (1995)
9. Bouyer, P., Forejt, V.: Reachability in stochastic timed games. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 103–114. Springer, Heidelberg (2009)
10. Brázdil, T., Forejt, V., Krčál, J., Křetínský, J., Kučera, A.: Continuous-time stochastic games with time-bounded reachability. In: Proceedings of FST&TCS 2009. LIPIcs, vol. 4, pp. 61–72. Schloss Dagstuhl (2009)
11. Brázdil, T., Krčál, J., Křetínský, J., Kučera, A., Řehák, V.: Stochastic real-time games with qualitative timed automata objectives. Technical report FIMU-RS-2010-05, Faculty of Informatics, Masaryk University (2010)
12. Chen, T., Han, T., Katoen, J.-P., Mereacre, A.: Quantitative model checking of continuous-time Markov chains against timed automata specifications. In: Proceedings of LICS 2009, pp. 309–318. IEEE, Los Alamitos (2009)
13. Haas, P.J., Shedler, G.S.: Regenerative generalized semi-Markov processes. Stochastic Models 3(3), 409–438 (1987)
14. Maitra, A., Sudderth, W.: Finitely additive stochastic games with Borel measurable payoffs. Int. Jour. of Game Theory 27, 257–267 (1998)
15. Martin, D.A.: The determinacy of Blackwell games. Journal of Symbolic Logic 63(4), 1565–1581 (1998)
16. Neuhäußer, M., Stoelinga, M., Katoen, J.-P.: Delayed nondeterminism in continuous-time Markov decision processes. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 364–379. Springer, Heidelberg (2009)
17. Norris, J.R.: Markov Chains. Cambridge University Press, Cambridge (1998)
18. Rabe, M., Schewe, S.: Optimal time-abstract schedulers for CTMDPs and Markov games. In: Eighth Workshop on Quantitative Aspects of Programming Languages (2010)
19. Ross, S.M.: Stochastic Processes. Wiley, Chichester (1996)
20. Schassberger, R.: Insensitivity of steady-state distributions of generalized semi-Markov processes. Advances in Applied Probability 10, 836–851 (1978)
21. Whitt, W.: Continuity of generalized semi-Markov processes. Mathematics of Operations Research 5(4), 494–501 (1980)

Session Types as Intuitionistic Linear Propositions

Luís Caires[1] and Frank Pfenning[2]

[1] CITI and Departamento de Informática, FCT, Universidade Nova de Lisboa
[2] Department of Computer Science, Carnegie Mellon University

Abstract. Several type disciplines for π-calculi have been proposed in which linearity plays a key role, even if their precise relationship with pure linear logic is still not well understood. In this paper, we introduce a type system for the π-calculus that exactly corresponds to the standard sequent calculus proof system for dual intuitionistic linear logic. Our type system is based on a new interpretation of linear propositions as session types, and provides the first purely logical account of all (both shared and linear) features of session types. We show that our type discipline is useful from a programming perspective, and ensures session fidelity, absence of deadlocks, and a tight operational correspondence between π-calculus reductions and cut elimination steps.

1 Introduction

Linear logic has been intensively explored in the analysis of π-calculus models for communicating and mobile system, given its essential ability to deal with resources, effects, and non-interference. The fundamental way it provides for analyzing notions of sharing versus uniqueness, captured by the exponential "!", seems to have been a source of inspiration for Milner when introducing replication in the π-calculus [22]. Following the early works of Abramsky [1], several authors have exploited variants of π-calculi to express proof reductions (*e.g.*, [5]) or game semantics (*e.g.*, [19]) in systems of linear logic. In the field of concurrency, many research directions have also drawn inspiration from linear logic for developing type-theoretic analyses of mobile processes, motivated by the works of Kobayashi, Pierce, and Turner [21]; a similar influence is already noticeable in the first publications by Honda on session types [16]. Many expressive type disciplines for π-calculi in which linearity frequently plays a key role have been proposed since then (*e.g.*, [20,18,26,15]). However, linearity has been usually employed in such systems in indirect ways, exploiting the fine grained type context management techniques it provides, or the assignment of usage multiplicities to channels [21], rather than the deeper type-theoretic significance of linear logical operators.

In this paper we present a type system for the π-calculus that exactly corresponds to the standard sequent calculus proof system for dual intuitionistic linear logic. The key to our correspondence is a new, perhaps surprising, interpretation of intuitionistic linear logic formulas as a form of session types [16,18], in which the programming language is a session-typed π-calculus, and the type structure consists precisely of the connectives of intuitionistic linear logic, retaining their standard proof-theoretic interpretation.

In session-based concurrency, processes communicate through so-called session channels, connecting exactly two subsystems, and communication is disciplined by session protocols so that actions always occur in dual pairs: when one partner sends, the

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 222–236, 2010.

other receives; when one partner offers a selection, the other chooses; when a session terminates, no further interaction may occur. New sessions may be dynamically created by invocation of shared servers. Such a model exhibits concurrency in the sense that several sessions, not necessarily causally related, may be executing simultaneously, although races in unshared resources are forbidden; in fact this is the common situation in disciplined concurrent programming idioms. Mobility is also present, since both session and server names may be passed around (delegated) in communications. Session types have been introduced to discipline interactions in session-based concurrency, an important paradigm in communication-centric programming (see [11]).

It turns out that the connectives of intuitionistic linear logic suffice to express all the essential features of finite session disciplines. While in the linear λ-calculus types are assigned to terms (denoting functions and values), in our interpretation types are assigned to names (denoting communication channels) and describe their session protocol. The essence of our interpretation may already be found in the interpretation of the linear logic multiplicatives as behavioral prefix operators. Traditionally, an object of type $A \multimap B$ denotes a linear function that given an object of type A returns an object of type B [14]. In our interpretation, an object of type $A \multimap B$ denotes a session x that first inputs a session channel of type A, and then behaves as B, where B specifies again an interactive behavior, rather than a closed value. Linearity of $\multimap$ is essential, otherwise the behavior of the input session after communication could not be ensured. An object of type $A \otimes B$ denotes a session that first sends a session channel of type A and afterwards behaves as B. But notice that objects of type $A \otimes B$ really consist of two objects: the sent session of type A and the continuation session, of type B. These two sessions are separate and non-interfering, as enforced by the canonical semantics of the linear multiplicative conjunction ($\otimes$). Our interpretation of $A \otimes B$ appears asymmetric, in the sense that, of course, a channel of type $A \otimes B$ is in general not typable by $B \otimes A$. In fact, the symmetry captured by the proof of $A \otimes B \vdash B \otimes A$ is realized by an appropriately typed process that coerces any session of type $A \otimes B$ to a session of type $B \otimes A$. The other linear constructors are also given compatible interpretations, in particular, the $!A$ type is naturally interpreted as a type of a shared server for sessions of type A, and additive product and sum, to branch and choice session type operators. We thus obtain the first purely logical account of both shared and linear features of session types.

We briefly summarize the contributions of the paper. We describe a system of session types for the π-calculus (Section 3) that corresponds to the sequent calculus for dual intuitionistic linear logic DILL (Section 4). The correspondence is bidirectional and tight, in the sense that (a) any π-calculus computation can be simulated by proof reductions on typing derivations (Theorem 5.3), thus establishing a strong form of subject reduction (Theorem 5.6), and (b) that any proof reduction or conversion corresponds either to a computation step or to a process equivalence on the π-calculus side (Theorems 5.4 and 5.5). An intrinsic consequence of the logical typing is a global progress property, that ensures the absence of deadlock for systems with an arbitrary number of open sessions (Theorem 5.8). Finally, we illustrate the expressiveness of our system (Section 6) with some examples and discussion.

2 Process Model

We briefly introduce the syntax and operational semantics of the process model: the synchronous π-calculus (see [24]) extended with (binary) guarded choice.

Definition 2.1 (Processes). *Given an infinite set Λ of* names (x, y, z, u, v)*, the set of* processes (P, Q, R) *is defined by*

$$\begin{array}{rlllll} P ::= \mathbf{0} \mid P \mid Q \mid (\boldsymbol{\nu} y)P & \mid x\langle y\rangle.P & \mid x(y).P & \mid\ !x(y).P \\ & \mid x.\mathtt{inl};P & \mid x.\mathtt{inr};P & \mid x.\mathtt{case}(P,Q) \end{array}$$

The operators $\mathbf{0}$ (inaction), $P \mid Q$ (parallel composition), and $(\boldsymbol{\nu} y)P$ (name restriction) comprise the static fragment of any π-calculus. We then have $x\langle y\rangle.P$ (send y on x and proceeds as P), $x(y).P$ (receive a name z on x and proceed as P with the input parameter y replaced by z), and $!x(y).P$ which denotes replicated (or persistent) input. The remaining three operators define a minimal labeled choice mechanism, comparable to the n-ary branching constructs found in standard session π-calculi (see eg., [18]). For the sake of minimality and without loss of generality we restrict our model to binary choice. In restriction $(\boldsymbol{\nu} y)P$ and input $x(y).P$ the distinguished occurrence of the name y is binding, with scope the process P. For any process P, we denote the set of *free names* of P by $\mathit{fn}(P)$. A process is *closed* if it does not contain free occurrences of names. We identify process up to consistent renaming of bound names, writing $\equiv_\alpha$ for this congruence. We write $P\{x/y\}$ for the process obtained from P by capture avoiding substitution of x for y in P. Structural congruence expresses basic identities on the structure of processes, while reduction expresses the behavior of processes.

Definition 2.2. Structural congruence *$(P \equiv Q)$, is the least congruence relation on processes such that*

$$\begin{array}{llll} P \mid \mathbf{0} \equiv P & \textit{(S}\mathbf{0}\textit{)} & P \equiv_\alpha Q \Rightarrow P \equiv Q & \textit{(S}\alpha\textit{)} \\ P \mid Q \equiv Q \mid P & \textit{(S|C)} & P \mid (Q \mid R) \equiv (P \mid Q) \mid R & \textit{(S|A)} \\ (\boldsymbol{\nu} x)\mathbf{0} \equiv \mathbf{0} & \textit{(S}\boldsymbol{\nu}\mathbf{0}\textit{)} & x \notin \mathit{fn}(P) \Rightarrow P \mid (\boldsymbol{\nu} x)Q \equiv (\boldsymbol{\nu} x)(P \mid Q) & \textit{(S}\boldsymbol{\nu}\textit{|)} \\ (\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P \equiv (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P & \textit{(S}\boldsymbol{\nu\nu}\textit{)} & & \end{array}$$

Definition 2.3. Reduction *$(P \to Q)$, is the binary relation on processes defined by:*

$$\begin{array}{ll} x\langle y\rangle.Q \mid x(z).P \to Q \mid P\{y/z\} & \textit{(RC)} \\ x\langle y\rangle.Q \mid\ !x(z).P \to Q \mid P\{y/z\} \mid\ !x(z).P & \textit{(R!)} \\ x.\mathtt{inl};P \mid x.\mathtt{case}(Q,R) \to P \mid Q & \textit{(RL)} \\ x.\mathtt{inr};P \mid x.\mathtt{case}(Q,R) \to P \mid R & \textit{(RR)} \\ Q \to Q' \Rightarrow P \mid Q \to P \mid Q' & \textit{(R|)} \\ P \to Q \Rightarrow (\boldsymbol{\nu} y)P \to (\boldsymbol{\nu} y)Q & \textit{(R}\boldsymbol{\nu}\textit{)} \\ P \equiv P',\, P' \to Q',\, Q' \equiv Q \Rightarrow P \to Q & \textit{(R}\equiv\textit{)} \end{array}$$

Notice that reduction is closed (by definition) under structural congruence. Reduction specifies the computations a process performs on its own. To characterize the interactions a process may perform with its environment, we introduce a labeled transition system; the standard early transition system for the π-calculus [24] extended with appropriate labels and transition rules for the choice constructs. A transition $P \xrightarrow{\alpha} Q$

$$\frac{P \xrightarrow{\alpha} Q}{(\boldsymbol{\nu} y)P \xrightarrow{\alpha} (\boldsymbol{\nu} y)Q}(\text{res}) \qquad \frac{P \xrightarrow{\alpha} Q}{P \mid R \xrightarrow{\alpha} Q \mid R}(\mathsf{par}) \qquad \frac{P \xrightarrow{\overline{\alpha}} P' \quad Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}(\mathsf{com})$$

$$\frac{P \xrightarrow{\overline{(\boldsymbol{\nu} y)x\langle y\rangle}} P' \quad Q \xrightarrow{x(y)} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu} y)(P' \mid Q')}\ (\mathsf{close}) \qquad \frac{P \xrightarrow{\overline{x\langle y\rangle}} Q}{(\boldsymbol{\nu} y)P \xrightarrow{\overline{(\boldsymbol{\nu} y)x\langle y\rangle}} Q}\ (\mathsf{open}) \qquad x\langle y\rangle.P \xrightarrow{\overline{x\langle y\rangle}} P\ (\mathsf{out})$$

$$x(y).P \xrightarrow{x(z)} P\{z/y\}\ (\mathsf{in}) \quad !x(y).P \xrightarrow{x(z)} P\{z/y\} \mid !x(y).P\ (\mathsf{rep}) \quad x.\mathtt{inl};P \xrightarrow{\overline{x.\mathtt{inl}}} P\ (\mathsf{lout})$$

$$x.\mathtt{inr};P \xrightarrow{\overline{x.\mathtt{inr}}} P\ (\mathsf{rout}) \quad x.\mathsf{case}(P,Q) \xrightarrow{x.\mathtt{inl}} P\ (\mathsf{lin}) \quad x.\mathsf{case}(P,Q) \xrightarrow{x.\mathtt{inr}} Q\ (\mathsf{rin})$$

Fig. 1. π-calculus Labeled Transition System

denotes that process P may evolve to process Q by performing the action represented by the label α. Transition labels are given by

$$\alpha ::= \overline{x\langle y\rangle} \mid x(y) \mid \overline{(\nu y)x\langle y\rangle} \mid x.\mathsf{inl} \mid x.\mathsf{inr} \mid \overline{x.\mathsf{inl}} \mid \overline{x.\mathsf{inr}}$$

Actions are input $x(y)$, the left/right offers $x.\mathtt{inl}$ and $x.\mathtt{inr}$, and their matching co-actions, respectively the output $\overline{x\langle y\rangle}$ and bound output $\overline{(\boldsymbol{\nu} y)x\langle y\rangle}$ actions, and the left/right selections $\overline{x.\mathtt{inl}}$ and $\overline{x.\mathtt{inr}}$. The bound output $\overline{(\boldsymbol{\nu} y)x\langle y\rangle}$ denotes extrusion of a fresh name y along (channel) x. Internal action is denoted by $\boldsymbol{\tau}$, in general an action α ($\overline{\alpha}$) requires a matching $\overline{\alpha}$ (α) in the environment to enable progress, as specified by the transition rules. For a label α, we define the sets $\mathit{fn}(\alpha)$ and $\mathit{bn}(\alpha)$ of free and bound names, respectively, as usual. We denote by $s(\alpha)$ the subject of α (*e.g.*, x in $x\langle y\rangle$).

Definition 2.4 (Labeled Transition System). *The relation* labeled transition *($P \xrightarrow{\alpha} Q$) is defined by the rules in Figure 1, subject to the side conditions: in rule* (res)*, we require $y \notin \mathit{fn}(\alpha)$; in rule* (par)*, we require $\mathit{bn}(\alpha) \cap \mathit{fn}(R) = \emptyset$; in rule* (close)*, we require $y \notin \mathit{fn}(Q)$. We omit the symmetric versions of rules* (par)*,* (com)*, and* (close).

We recall some basic facts about reduction, structural congruence, and labeled transition, namely: closure of labeled transitions under structural congruence, and coincidence of $\boldsymbol{\tau}$-labeled transition and reduction [24]: (1) if $P \equiv\xrightarrow{\alpha} Q$, then $P \xrightarrow{\alpha}\equiv Q$, and (2) $P \to Q$ if and only if $P \xrightarrow{\boldsymbol{\tau}}\equiv Q$. We write $\rho_1\rho_2$ for relation composition (e.g, $\xrightarrow{\boldsymbol{\tau}}\equiv$).

3 Type System

We first describe our type structure, which coincides with intuitionistic linear logic [14,3], omitting atomic formulas and the additive constants $\top$ and $\mathbf{0}$.

Definition 3.1 (Types). *Types (A, B, C) are given by*

$$A, B ::= \mathbf{1} \mid !A \mid A \otimes B \mid A \multimap B \mid A \oplus B \mid A \mathbin{\&} B$$

Types are assigned to (channel) names, and may be conveniently interpreted as a form of session types; an assignment $x{:}A$ enforces that the process will use x according to the discipline A. $A \otimes B$ is the type of a session channel that first performs an output

$$\frac{\Gamma; \Delta \vdash P :: T}{\Gamma; \Delta, x{:}\mathbf{1} \vdash P :: T}\ (\mathrm{T1L}) \qquad \frac{}{\Gamma; \cdot \vdash 0 :: x{:}\mathbf{1}}\ (\mathrm{T1R})$$

$$\frac{\Gamma; \Delta, y{:}A, x{:}B \vdash P :: T}{\Gamma; \Delta, x{:}A \otimes B \vdash x(y).P :: T}\ (\mathrm{T}{\otimes}\mathrm{L}) \qquad \frac{\Gamma; \Delta \vdash P :: y{:}A \quad \Gamma; \Delta' \vdash Q :: x{:}B}{\Gamma; \Delta, \Delta' \vdash (\nu y)x\langle y\rangle.(P \mid Q) :: x{:}A \otimes B}\ (\mathrm{T}{\otimes}\mathrm{R})$$

$$\frac{\Gamma; \Delta \vdash P :: y{:}A \quad \Gamma; \Delta', x{:}B \vdash Q :: T}{\Gamma; \Delta, \Delta', x{:}A \multimap B \vdash (\nu y)x\langle y\rangle.(P \mid Q) :: T}\ (\mathrm{T}{\multimap}\mathrm{L}) \qquad \frac{\Gamma; \Delta, y{:}A \vdash P :: x{:}B}{\Gamma; \Delta \vdash x(y).P :: x{:}A \multimap B}\ (\mathrm{T}{\multimap}\mathrm{R})$$

$$\frac{\Gamma; \Delta \vdash P :: x{:}A \quad \Gamma; \Delta', x{:}A \vdash Q :: T}{\Gamma; \Delta, \Delta' \vdash (\nu x)(P \mid Q) :: T}\ (\mathsf{Tcut}) \qquad \frac{\Gamma; \cdot \vdash P :: y{:}A \quad \Gamma, u{:}A; \Delta \vdash Q :: T}{\Gamma; \Delta \vdash (\nu u)(!u(y).P \mid Q) :: T}\ (\mathsf{Tcut}^{!})$$

$$\frac{\Gamma, u{:}A; \Delta, y{:}A \vdash P :: T}{\Gamma, u{:}A; \Delta \vdash (\nu y)u\langle y\rangle.P :: T}\ (\mathsf{Tcopy})$$

$$\frac{\Gamma, u{:}A; \Delta \vdash P\{u/x\} :: T}{\Gamma; \Delta, x{:}!A \vdash P :: T}\ (\mathrm{T!L}) \qquad \frac{\Gamma; \cdot \vdash Q :: y{:}A}{\Gamma; \cdot \vdash !x(y).Q :: x{:}!A}\ (\mathrm{T!R})$$

$$\frac{\Gamma; \Delta, x{:}A \vdash P :: T \quad \Gamma; \Delta, x{:}B \vdash Q :: T}{\Gamma; \Delta, x{:}A \oplus B \vdash x.\mathtt{case}(P, Q) :: T}\ (\mathrm{T}{\oplus}\mathrm{L}) \qquad \frac{\Gamma; \Delta, x{:}B \vdash P :: T}{\Gamma; \Delta, x{:}A \,\&\, B \vdash x.\mathtt{inr}; P :: T}\ (\mathrm{T}\&\mathrm{L}_2)$$

$$\frac{\Gamma; \Delta \vdash P :: x{:}A \quad \Gamma; \Delta \vdash Q :: x{:}B}{\Gamma; \Delta \vdash x.\mathtt{case}(P, Q) :: x{:}A \,\&\, B}\ (\mathrm{T}\&\mathrm{R}) \qquad \frac{\Gamma; \Delta, x{:}A \vdash P :: T}{\Gamma; \Delta, x{:}A \,\&\, B \vdash x.\mathtt{inl}; P :: T}\ (\mathrm{T}\&\mathrm{L}_1)$$

$$\frac{\Gamma; \Delta \vdash P :: x{:}A}{\Gamma; \Delta \vdash x.\mathtt{inl}; P :: x{:}A \oplus B}\ (\mathrm{T}{\oplus}\mathrm{R}_1) \qquad \frac{\Gamma; \Delta \vdash P :: x{:}B}{\Gamma; \Delta \vdash x.\mathtt{inr}; P :: x{:}A \oplus B}\ (\mathrm{T}{\oplus}\mathrm{R}_2)$$

Fig. 2. The Type System πDILL

(sending a session channel of type A) to its partner before proceeding as specified by B. In a similar way, $A \multimap B$ types a session channel that first performs an input (receiving a session channel of type A) from its partner, before proceeding as specified by B. The type $\mathbf{1}$ means that the session terminated, no further interaction will take place on it. Notice that names of type $\mathbf{1}$ may still be passed around in sessions, as opaque values. $A \oplus B$ types a session that either selects "left" and then proceed as specified by A, or else selects "right", and then proceeds as specified by B. Dually, $A \& B$ types a session channel that offers its partner a choice between an A typed behavior ("left" choice) and a B typed behavior ("right" choice). The type $!A$ types a non-session (non-linearized, shared) channel (called *standard channel* in [13]), to be used by a server for spawning an arbitrary number of new sessions (possibly none), each one conforming to type A.

A type environment is a collection of type assignments, of the form $x : A$ where x is a name and A a type, the names being pairwise disjoint. Following the insights behind dual intuitionistic linear logic, which goes back to Andreoli's *dyadic* system for classical linear logic [2], we distinguish two kinds of type environments subject to different structural properties: a *linear* part Δ and an *unrestricted* part Γ, where weakening and contraction principles hold for Γ but not for Δ. A judgment of our system has then the form $\Gamma; \Delta \vdash P :: z{:}C$ where name declarations in Γ are always propagated unchanged to all premises in the typing rules, while name declarations in Δ are handled multiplicatively or additively, depending on the nature of the type being defined. The domains of Γ, Δ and $z{:}C$ are required to be pairwise disjoint.

Intuitively, such a judgment asserts: P is ensured to safely provide a usage of name z according to the behavior (session) specified by type C, whenever composed with any process environment providing usages of names according to the behaviors specified by names in $\Gamma; \Delta$. As shown in Section 5, in our case safety ensures that the behavior is free of communication errors and deadlock. A pure client Q that just relies on external services, and does not provide any, will be typed as $\Gamma; \Delta \vdash Q :: -{:}\mathbf{1}$. In general, a process P such that $\Gamma; \Delta \vdash P :: z{:}C$ represents a system providing behavior C at channel z, building on "services" declared in $\Gamma; \Delta$. Of particular interest is a system typed as $\Gamma; \Delta \vdash R :: z{:}!A$, representing a shared server. Quite interestingly, the asymmetry induced by the intuitionistic interpretation of $!A$ enforces locality of shared names but not of linear (session names), which exactly corresponds to the intended model of sessions.

We present the rules of our type system $\pi\mathtt{DILL}$ in Fig. 2. We use T, S for right hand side singleton environments (e.g., $z{:}C$). The interpretation of the various rules should be clear, given the explanation of types given above. Notice that since in $\otimes$R the sent name is always fresh, our typed calculus conforms to a session-based internal mobility discipline [23,7], without loss of expressiveness. The composition rules (cut and cut$^!$) follow the "composition plus hiding" principle [1], extended to a name passing setting. More familiar linear typing rules for parallel composition (*e.g.*, as in [21]) are derivable (see Section 6). Since we are considering π-calculus terms up to structural congruence, typability is closed under $\equiv$ by definition. $\pi\mathtt{DILL}$ enjoys the usual properties of equivariance, weakening in Γ and contraction in Γ. The coverage property also holds: if $\Gamma; \Delta \vdash P :: z{:}A$ then $\mathit{fn}(P) \subseteq \Gamma \cup \Delta \cup \{z\}$. In the presence of type-annotated restrictions $(\nu x{:}A)P$, as usual in typed π-calculi [24], type-checking is decidable.

We illustrate the type system with a simple example, frequently used to motivate session based interactions (see *e.g.*, [13]). A client may choose between a "buy" operation, in which it indicates a product name and a credit card number to receive a receipt, and a "quote" operation, in which it indicates a product name, to obtain the product price. From the client perspective, the session protocol exposed by the server may be specified by the type

$$\mathit{ServerProto} \triangleq (N \multimap I \multimap (N \otimes \mathbf{1}))\ \&\ (N \multimap (I \otimes \mathbf{1}))$$

We assume that N and I are types representing shareable values (*e.g.*, strings N and integers I). To simplify, we set $N = I = \mathbf{1}$. Assuming s to be the name of the session channel connecting the client and server, consider the code

$$\mathit{QClntBody}_s \triangleq s.\mathtt{inr}; (\boldsymbol{\nu} \mathit{tea}) s\langle \mathit{tea} \rangle . s(\mathit{pr}).\mathbf{0}$$

$\mathit{QClntBody}_s$ specifies a client that asks for the price of tea (we simply abstract away from what the client might do with the price after reading it). It first selects the quoting operation on the server ($s.\mathtt{inr}$), then sends the id of the product to the server ($s\langle \mathit{tea} \rangle$), then receives the price $s(\mathit{pr})$ from the server and finally terminates the session ($\mathbf{0}$). Then

$$\cdot; s : \mathit{ServerProto} \vdash \mathit{QClntBody}_s :: -{:}\mathbf{1}$$

is derivable (by $T\mathbf{1}$R, $T\otimes$L, $T\multimap$L and $T\&$L$_2$). Here we wrote $-$ for an anonymous variable that does not appear in *QClntBody*. This is possible even in a linear type discipline since the inactive process $\mathbf{0}$ is typed by $x{:}\mathbf{1}$ and does not use x. Concerning the server code, let $\mathit{SrvBody}_s \triangleq s.\mathtt{case}(\ s(\mathit{pn}).s(\mathit{cn}).(\nu \mathit{rc})s\langle \mathit{rc} \rangle.\mathbf{0},\ s(\mathit{pn}).(\nu \mathit{pr})s\langle \mathit{pr} \rangle.\mathbf{0}\ )$.

$$\frac{\Gamma; \Delta \vdash D : C}{\Gamma; \Delta, x : \mathbf{1} \vdash \mathsf{1L}\ x\ D : C}\ (\mathsf{1L}) \qquad \frac{}{\Gamma; \cdot \vdash \mathsf{1R} : \mathbf{1}}\ (\mathsf{1R})$$

$$\frac{\Gamma; \Delta, y : A, x : B \vdash D : C}{\Gamma; \Delta, x : A \otimes B \vdash \otimes\mathsf{L}\ x\ (y.x.\ D) : C}\ (\otimes\mathsf{L}) \qquad \frac{\Gamma; \Delta \vdash D : A \quad \Gamma; \Delta' \vdash E : B}{\Gamma; \Delta, \Delta' \vdash \otimes\mathsf{R}\ D\ E : A \otimes B}\ (\otimes\mathsf{R})$$

$$\frac{\Gamma; \Delta \vdash D : A \qquad \Gamma; \Delta', x : B \vdash E : C}{\Gamma; \Delta, \Delta', x : A \multimap B \vdash \multimap\mathsf{L}\ x\ D\ (x.\ E) : C}\ (\multimap\mathsf{L}) \qquad \frac{\Gamma; \Delta, y : A \vdash D : B}{\Gamma; \Delta \vdash \multimap\mathsf{R}\ (y.\ D) : A \multimap B}\ (\multimap\mathsf{R})$$

$$\frac{\Gamma; \Delta \vdash D : A \quad \Gamma; \Delta', x : A \vdash E : C}{\Gamma; \Delta, \Delta' \vdash \mathsf{cut}\ D\ (x.E) : C}\ (\mathsf{cut}) \qquad \frac{\Gamma; \cdot \vdash D : A \quad \Gamma, u : A; \Delta \vdash E : C}{\Gamma; \Delta \vdash \mathsf{cut}^{!}\ D\ (u.\ E) : C}\ (\mathsf{cut}^{!})$$

$$\frac{\Gamma, u : A; \Delta, y : A \vdash D : C}{\Gamma, u : A; \Delta \vdash \mathsf{copy}\ u\ (y.\ D) : C}\ (\mathsf{copy})$$

$$\frac{\Gamma; \cdot \vdash D : A}{\Gamma; \cdot \vdash \mathsf{!R}\ D : !A}\ (\mathsf{!R}) \qquad \frac{\Gamma, u : A; \Delta \vdash D : C}{\Gamma; \Delta, x : !A \vdash \mathsf{!L}\ x\ (u.D) : C}\ (\mathsf{!L})$$

$$\frac{\Gamma; \Delta, x : A \vdash D : C}{\Gamma; \Delta, x : A \mathbin{\&} B \vdash \&\mathsf{L}_1\ x\ (x.\ D) : C}\ (\&\mathsf{L}_1) \qquad \frac{\Gamma; \Delta, x : B \vdash D : C}{\Gamma; \Delta, x : A \mathbin{\&} B \vdash \&\mathsf{L}_2\ x\ (x.\ D) : C}\ (\&\mathsf{L}_2)$$

$$\frac{\Gamma; \Delta \vdash D : A \quad \Gamma; \Delta \vdash E : B}{\Gamma; \Delta \vdash \&\mathsf{R}\ D\ E : A \mathbin{\&} B}\ (\&\mathsf{R}) \qquad \frac{\Gamma; \Delta x : A \vdash D : C \quad \Gamma; \Delta, x : B \vdash E : C}{\Gamma; \Delta, x : A \oplus B \vdash \oplus\mathsf{L}\ x\ (x.\ D)\ (x.\ E) : C}\ (\oplus\mathsf{L})$$

$$\frac{\Gamma; \Delta \vdash D : A}{\Gamma; \Delta \vdash \oplus\mathsf{R}_1\ D : A \oplus B}\ (\oplus\mathsf{R}_1) \qquad \frac{\Gamma; \Delta \vdash D : B}{\Gamma; \Delta \vdash \oplus\mathsf{R}_2\ D : A \oplus B}\ (\oplus\mathsf{R}_2)$$

Fig. 3. Dual Intuitionistic Linear Logic DILL

Then $\cdot; \cdot \vdash \mathit{SrvBody}_s :: s{:}\mathit{ServerProto}$ is derivable, by $T\&\mathsf{R}$. By $T\mathsf{cut}$ we obtain for the system $\mathit{QSimple} \triangleq (\boldsymbol{\nu} s)(\mathit{SrvBody}_s \mid \mathit{QClntBody}_s)$ the typing $\cdot; \cdot \vdash \mathit{QSimple} :: -{:}\mathbf{1}$. In this example we have only introduced processes interacting in a single session, but clearly the system accomodates all the generality of session types, e.g., a simple process interacting in different sessions is $x{:}A \multimap \mathbf{1}, y{:}A \otimes \mathbf{1} \vdash y(w).(\boldsymbol{\nu} k)x\langle k\rangle.\mathbf{0} :: -{:}\mathbf{1}$.

4 Dual Intuitionistic Linear Logic

As presented, session type constructors correspond directly to intuitionistic linear logic connectives. Typing judgments directly correspond to sequents in dual intuitionistic linear logic, by erasing processes [3,10]. In Figure 3 we present the DILL sequent calculus. In our presentation, DILL is conveniently equipped with a faithful proof term assignment, so sequents have the form $\Gamma; \Delta \vdash D : C$ where Γ is the unrestricted context, Δ the linear context, C a formula (= type) and D the proof term that faithfully represents the derivation of $\Gamma; \Delta \vdash C$. Our use of names in the proof system will be consistent with the proof discipline, u, v, w for variables in Γ and x, y, z for variables in Δ. This is consistent with standard usage of names in π-calculi. Given the parallel structure of the two systems, if $\Gamma; \Delta \vdash D : A$ is derivable in DILL then there is a process P and a name z such that $\Gamma; \Delta \vdash P :: z{:}A$ is derivable in πDILL, and the converse result also holds: if $\Gamma; \Delta \vdash P :: z{:}A$ is derivable in πDILL there is a

D	$\rightsquigarrow \hat{D}^z$	D	$\rightsquigarrow \hat{D}^z$
$\mathsf{1R}$	$\rightsquigarrow 0$	$\oplus\mathsf{R}_1\ D$	$\rightsquigarrow z.\mathsf{inl}; \hat{D}^z$
$\mathsf{1L}\ x\ D$	$\rightsquigarrow \hat{D}^z$	$\oplus\mathsf{R}_2\ E$	$\rightsquigarrow z.\mathsf{inr}; \hat{E}^z$
$\otimes\mathsf{R}\ D\ E$	$\rightsquigarrow (\nu y)\, z\langle y\rangle.\, (\hat{D}^y \mid \hat{E}^z)$	$\oplus\mathsf{L}\ x\ (x.\, D)\ (x.\, E)$	$\rightsquigarrow x.\mathsf{case}(\hat{D}^z, \hat{E}^z)$
$\otimes\mathsf{L}\ x\ (y.x.\, D)$	$\rightsquigarrow x(y).\, \hat{D}^z$	$\mathsf{cut}\ D\ (x.\, E)$	$\rightsquigarrow (\nu x)(\hat{D}^x \mid \hat{E}^z)$
$\multimap\mathsf{R}\ (y.\, D)$	$\rightsquigarrow z(y).\, \hat{D}^z$		
$\multimap\mathsf{L}\ x\ D\ (x.\, E)$	$\rightsquigarrow (\nu y)\, x\langle y\rangle.\, (\hat{D}^y \mid \hat{E}^z)$	$!\mathsf{R}\ D$	$\rightsquigarrow\ !z(y).\, \hat{D}^y$
$\&\mathsf{R}\ D\ E$	$\rightsquigarrow z.\, \mathsf{case}(\hat{D}^z, \hat{E}^z)$	$!\mathsf{L}\ x\ (u.\, D)$	$\rightsquigarrow \hat{D}^z\{x/u\}$
$\&\mathsf{L}_1\ x\ (x.\, D)$	$\rightsquigarrow x.\, \mathsf{inl}; \hat{D}^z$	$\mathsf{copy}\ u\ (y.\, D)$	$\rightsquigarrow (\nu y)\, u\langle y\rangle.\, \hat{D}^z$
$\&\mathsf{L}_2\ x\ (x.\, E)$	$\rightsquigarrow x.\, \mathsf{inr}; \hat{E}^z$	$\mathsf{cut}^!\ D\ (u.\, E)$	$\rightsquigarrow (\nu u)((!u(y).\, \hat{D}^y) \mid \hat{E}^z)$

Fig. 4. Proof D extracts to process $\hat{D}^z$

derivation D that proves $\Gamma; \Delta \vdash D : A$. This correspondence is made explicit by a translation from faithful proof terms to processes, defined in Fig. 4: for $\Gamma; \Delta \vdash D : C$ we write $\hat{D}^z$ for the translation of D such that $\Gamma; \Delta \vdash \hat{D}^z :: z{:}C$.

Definition 4.1 (Typed Extraction). *We write $\Gamma; \Delta \vdash D \rightsquigarrow P :: z{:}A$, meaning "proof D extracts to P", whenever $\Gamma; \Delta \vdash D : A$ and $\Gamma; \Delta \vdash P :: z{:}A$ and $P \equiv \hat{D}^z$.*

Typed extraction is unique up to structural congruence, in the sense that if $\Gamma; \Delta \vdash D \rightsquigarrow P :: z{:}A$ and $\Gamma; \Delta \vdash D \rightsquigarrow Q :: z{:}A$ then $P \equiv Q$, as a consequence of closure of typing under structural congruence. The system DILL as presented does not admit atomic formulas, and hence has no true initial sequents. However, the correspondence mentioned above yields an explicit identity theorem:

Proposition 4.2. *For any type A and distinct names x, y, there is a process $id_A(x, y)$ and a cut-free derivation D such that $\cdot; x{:}A \vdash D \rightsquigarrow id_A(x, y) :: y{:}A$.*

The $id_A(x, y)$ process, with exactly the free names x, y, implements a synchronous mediator that bidirectionally plays the protocol specified by A between channels x and y. For example, we analyze the interpretation of the sequent $A \otimes B \vdash B \otimes A$. We have

$$x{:}A \otimes B \vdash F \rightsquigarrow x(z).(\nu n)y\langle n\rangle.(P \mid Q) :: y{:}B \otimes A$$

where $F = \otimes\mathsf{L}\ x\ (z.x.\ \otimes\mathsf{R}\ D\ E)$, $P = id_B(x, n)$ and $Q = id_A(z, y)$. This process is an interactive proxy that coerces a session of type $A \otimes B$ at x to a session of type $B \otimes A$ at y. It first receives a session of type A (bound to z) and after sending on y a session of type B (played by copying the continuation of x to n), it progresses with a session of type A on y (copying the continuation of z to y).

As processes are related by structural and computational rules, namely those involved in the definition of $\equiv$ and $\rightarrow$, derivations in DILL are related by structural and computational rules, that express certain sound proof transformations that arise in cut-elimination. The reductions (Figure 5) generally take place when a right rule meets a left rule for the same connective, and correspond to reduction steps in the process term assignment. On the left, we show the usual reductions for cuts; on the right, we show the corresponding reductions (if any) of the process terms, modulo structural congruence. Since equivalences depend on variable occurrences, we write D_x if x may occur in D.

The structural conversions in Figure 6 correspond to structural equivalences in the π-calculus, since they just change the order of cuts, *e.g.*, $(\mathsf{cut}/-/\mathsf{cut}_1)$ translates to

$$(\nu x)(\hat{D}^x \mid (\nu y)(\hat{E}^y \mid \hat{F}^z)) \equiv (\nu y)((\nu x)(\hat{D}^x \mid \hat{E}^y) \mid \hat{F}^z)$$

In addition, we have two special conversions. Among those, $(\mathsf{cut}/\mathbf{1}\mathsf{R}/\mathbf{1}\mathsf{L})$ is not needed in order to simulate the π-calculus reduction, while $(\mathsf{cut}/!\mathsf{R}/!\mathsf{L})$ is. In cut-elimination procedures, these are always used from left to right. Here, they are listed as equivalences because the corresponding π-calculus terms are structurally congruent. The root cause for this is that the rules $\mathbf{1}\mathsf{L}$ and $!\mathsf{L}$ are *silent*: the extracted terms in the premise and conclusion are the same, modulo renaming. For $\mathbf{1}\mathsf{L}$, this is the case because a terminated process, represented by $0 :: - : \mathbf{1}$ silently disappears from a parallel composition by structural congruence. For $!\mathsf{L}$, this is the case because the actual replication of a server process is captured in the copy rule which clones $u{:}A$ to $y{:}A$, rule rather than $!\mathsf{L}$. It is precisely for this reason that the rule commuting a persistent cut $(\mathsf{cut}^!)$ over a copy rule (copy) is among the computational conversions.

The structural conversions in Figure 8 propagate $\mathsf{cut}^!$. From the proof theoretic perspective, because $\mathsf{cut}^!$ cuts a persistent variable u, $\mathsf{cut}^!$ may be duplicated or erased. On the π-calculus side, these no longer correspond to structural congruences, but, quite remarkably, to behavioral equivalences, derivable from known properties of typed processes, the (sharpened) Replication Theorems [24]. These hold in our language, due to our interpretation of ! types. Our operational correspondence results also depend on six commuting conversions, four in Figure 7 plus two symmetric versions. The commuting conversions push a cut up (or inside) the $\mathbf{1}\mathsf{L}$ and $!\mathsf{L}$ rules. During the usual cut elimination procedures, these are used from left to right. In the correspondence with the sequent calculus, the situation is more complex. Because the $\mathbf{1}\mathsf{L}$ and $!\mathsf{L}$ rules do not affect the extracted term, cuts have to be permuted with these two rules in order to simulate π-calculus reduction. From the process calculus perspective, such conversions correspond to identity. There is a second group of commuting conversions (not shown), not necessary for our current development. Those do not correspond to structural congruence nor to strong bisimilarities on π-calculus, as they may not preserve process behavior in the general untyped setting, since they promote an action prefix from a subexpression to the top level. We conjecture that such equations denote behavioral identities under a natural definition of typed observational congruence for our calculus.

Definition 4.3 (Relations on derivations induced by conversions). *(1) $\equiv$: the least congruence on derivations generated by the structural conversions (I) and the commuting conversions (II): (2) $\simeq_s$: the least congruence on derivations generated by all structural conversions (I-III). We extend $\simeq_s$ to processes as the congruence generated by the process equations on the right. (3) $\Rrightarrow$: the reduction on derivations obtained by orienting all conversions in the direction shown, from left to right or top to bottom.*

As discussed above, $\simeq_s$ is a typed behavioral equivalence on processes.

5 Computational Correspondence, Preservation, and Progress

We now present the results stating the key properties of our type system and logical interpretation. Theorem 5.3 states the existence of a simulation between reductions in

$$\begin{array}{ll}
\mathsf{cut}\ (\otimes\mathsf{R}\ D_1\ D_2)\ (x.\,\otimes\mathsf{L}\ x\ (y.x.\,E_{xy})) & \rightsquigarrow (\nu x)(((\nu y)\,x\langle y\rangle.\,(\hat{D}_1^y \mid \hat{D}_2^x)) \mid x(y).\,\hat{E}^z) \\
\Rightarrow & \longrightarrow \\
\mathsf{cut}\ D_1\ (y.\,\mathsf{cut}\ D_2\ (x.\,E_{xy})) & \rightsquigarrow (\nu x)(\nu y)(\hat{D}_1^y \mid \hat{D}_2^x \mid \hat{E}^z) \\
\mathsf{cut}\ (\multimap\mathsf{R}\ (y.\,D_y))\ (x.\,\multimap\mathsf{L}\ x\ E_1\ (x.\,E_{2x})) & \rightsquigarrow (\nu x)((x(y).\,\hat{D}^x) \mid (\nu y)\,x\langle y\rangle.\,(\hat{E}_1^y \mid \hat{E}_2^z)) \\
\Rightarrow & \longrightarrow \\
\mathsf{cut}\ (\mathsf{cut}\ E_1\ (y.\,D_y))\ (x.\,E_{2x}) & \rightsquigarrow (\nu x)(\nu y)(\hat{D}^x \mid \hat{E}_1^y \mid \hat{E}_2^z) \\
\mathsf{cut}\ (\&\mathsf{R}\ D_1\ D_2)\ (x.\&\mathsf{L}_i\ x\ (x.\,E_x)) & \rightsquigarrow (\nu x)(x.\mathsf{case}(\hat{D}_1^x, \hat{D}_2^x) \mid x.\mathsf{inl};\hat{E}^z) \\
\Rightarrow & \longrightarrow \\
\mathsf{cut}\ D_i\ (x.\,E_x) & \rightsquigarrow (\nu x)(\hat{D}_i^x \mid \hat{E}^z) \\
\mathsf{cut}\ (\oplus\mathsf{R}_i\ D)\ (x.\oplus\mathsf{L}\ x\ (x.\,E_{1x})\ (x.\,E_{2x})) & \rightsquigarrow (\nu x)(x.\mathsf{inl};\hat{D}^x \mid x.\mathsf{case}(\hat{E}_1^z, \hat{E}_2^z)) \\
\Rightarrow & \longrightarrow \\
\mathsf{cut}\ D\ (x.\,E_{ix}) & \rightsquigarrow (\nu x)(\hat{D}^x \mid \hat{E}_i^z) \\
\mathsf{cut}^!\ D\ (u.\,\mathsf{copy}\ u\ (y.\ E_{uy})) & \rightsquigarrow (\nu u)((!u(y).\,\hat{D}^y) \mid (\nu y)\,u\langle y\rangle.\,\hat{E}^z) \\
\Rightarrow & \longrightarrow \\
\mathsf{cut}\ D\ (y.\,\mathsf{cut}^!\ D\ (u.\ E_{uy})) & \rightsquigarrow (\nu y)(\hat{D}^y \mid (\nu u)((!u(y).\,\hat{D}^y) \mid \hat{E}^z))
\end{array}$$

Fig. 5. Computational Conversions

$$\begin{array}{lll}
(\mathsf{cut}/-/\mathsf{cut}_1) & \mathsf{cut}\ D\ (x.\,\mathsf{cut}\ E_x\ (y.\ F_y)) & \equiv \mathsf{cut}\ (\mathsf{cut}\ D\ (x.\,E_x))\ (y.\ F_y) \\
(\mathsf{cut}/-/\mathsf{cut}_2) & \mathsf{cut}\ D\ (x.\,\mathsf{cut}\ E\ (y.\ F_{xy})) & \equiv \mathsf{cut}\ E\ (y.\,\mathsf{cut}\ D\ (x.\ F_{xy})) \\
(\mathsf{cut}/\mathsf{cut}^!/-) & \mathsf{cut}\ (\mathsf{cut}^!\ D\ (u.\,E_u))\ (x.\,F_x) & \equiv \mathsf{cut}^!\ D\ (u.\,\mathsf{cut}\ E_u\ (x.\,F_x)) \\
(\mathsf{cut}/-/\mathsf{cut}^!) & \mathsf{cut}\ D\ (x.\,\mathsf{cut}^!\ E\ (u.\,F_{xu})) & \equiv (\mathsf{cut}^!\ E\ (u.\,\mathsf{cut}\ D\ (x.\,F_{xu})) \\
(\mathsf{cut}/\mathbf{1}\mathsf{R}/\mathbf{1}\mathsf{L}) & \mathsf{cut}\ \mathbf{1}\mathsf{R}\ (x.\,\mathbf{1}\mathsf{L}\ x\ D) & \equiv D \\
(\mathsf{cut}/!\mathsf{R}/!\mathsf{L}) & \mathsf{cut}\ (!\mathsf{R}\ D)\ (x.\,!\mathsf{L}\ x\ (u.\,E)) & \equiv \mathsf{cut}^!\ D\ (u.\,E)
\end{array}$$

Fig. 6. Structural Conversions (I): Cut Conversions

$$\begin{array}{lll}
(\mathsf{cut}/\mathbf{1}\mathsf{L}/-) & \mathsf{cut}\ (\mathbf{1}\mathsf{L}\ y\ D)\ (x.\,F_x) & \equiv \mathbf{1}\mathsf{L}\ y\ (\mathsf{cut}\ D\ (x.\,F_x)) \\
(\mathsf{cut}/!\mathsf{L}/-) & \mathsf{cut}\ (!\mathsf{L}\ y\ (u.\,D_u))\ (x.\,F_x) & \equiv\ !\mathsf{L}\ y\ (u.\,\mathsf{cut}\ D_u\ (x.\,F_x)) \\
(\mathsf{cut}^!/-/\mathbf{1}\mathsf{L}) & \mathsf{cut}^!\ D\ (u.\,\mathbf{1}\mathsf{L}\ y\ E_u) & \equiv \mathbf{1}\mathsf{L}\ y\ (\mathsf{cut}^!\ D\ (u.\,E_u)) \\
(\mathsf{cut}^!/-/!\mathsf{L}) & \mathsf{cut}^!\ D\ (u.\,!\mathsf{L}\ y\ (v.\,E_{uv})) & \equiv\ !\mathsf{L}\ y\ (v.\ \mathsf{cut}^!\ D\ (u.\,E_{uv}))
\end{array}$$

Fig. 7. Structural Conversions (II): Commuting Conversions

$$\begin{array}{ll}
\mathsf{cut}^!\ D\ (u.\,\mathsf{cut}\ E_u\ (y.\ F_{uy})) & \rightsquigarrow (\nu u)(!u(y).\hat{D}^y \mid (\nu y)(\hat{E}^y \mid \hat{F}^z)) \\
\simeq & \simeq \\
\mathsf{cut}\ (\mathsf{cut}^!\ D\ (u.\,E_u))\ (y.\ \mathsf{cut}^!\ D\ (u.\,F_{uy})) & \rightsquigarrow (\nu y)((\nu u)(!u(y).\hat{D}^y \mid \hat{E}^y) \mid \\
 & \qquad (\nu u)(!u(y).\hat{D}^y \mid \hat{F}^z)\ ) \\
\mathsf{cut}^!\ D\ (u.\,\mathsf{cut}^!\ E_u\ (v.\ F_{uv})) & \rightsquigarrow (\nu u)(!u(y).\hat{D}^y \mid (\nu v)(!v(y).\hat{E}^y \mid \hat{F}^z)) \\
\simeq & \simeq \\
\mathsf{cut}^!\ (\mathsf{cut}^!\ D\ (u.\,E_u))\ (v.\ \mathsf{cut}^!\ D\ (u.\,F_{uv})) & \rightsquigarrow (\nu v)((!v(y).(\nu u)(!u(y).\hat{D}^y \mid \hat{E}^y)) \mid \\
 & \qquad (\nu u)(!u(y).\hat{D}^y \mid \hat{F}^z)\ ) \\
\mathsf{cut}^!\ (\mathsf{cut}^!\ D\ (u.\,E_u))\ (v.\ F_v) & \rightsquigarrow (\nu v)(!v(y).(\nu u)(!u(y).\hat{D}^y \mid \hat{E}^y)) \mid F^z) \\
\simeq & \simeq \\
\mathsf{cut}^!\ D\ (u.\,\mathsf{cut}^!\ E_u\ (v.\ F_v)) & \rightsquigarrow (\nu u)(!u(y).\hat{D}^y \mid (\nu v)(!v(y).\hat{E}^y \mid \hat{F}^z)) \\
\mathsf{cut}^!\ D\ (u.\,E) & \rightsquigarrow (\nu u)(!u(y).\hat{D}^y \mid \hat{E}^z) \\
\simeq & \simeq \\
E & \rightsquigarrow \hat{E}^z \quad (\text{for } u \notin FN(\hat{E}^z))
\end{array}$$

Fig. 8. Structural Conversions (III): Cut! Conversions

the typed π-calculus and proof conversions / reductions, expressing a strong form of subject reduction for our type system. The proof relies on several auxiliary lemmas, which we mostly omit, among them a sequence of lemmas relating process reduction with derivation reduction, from which we select two typical examples.

Lemma 5.1. *Assume (a)* $\Gamma; \Delta_1 \vdash D \rightsquigarrow P :: x{:}A_1 \otimes A_2$ *with* $P \overset{\overline{(\nu y)x\langle y\rangle}}{\rightarrow} P'$*; and (b)* $\Gamma; \Delta_2, x{:}A_1 \otimes A_2 \vdash E \rightsquigarrow Q :: z{:}C$ *with* $Q \overset{x(y)}{\rightarrow} Q'$*. Then (c)* $\mathsf{cut}\ D\ (x.\,E) \equiv\Rightarrow\equiv F$ *for some* F*; (d)* $\Gamma; \Delta_1, \Delta_2 \vdash F \rightsquigarrow R :: z : C$ *for* $R \equiv (\nu y)(\nu x)(P' \mid Q')$.

Lemma 5.2. *Assume (a)* $\Gamma; \cdot \vdash D \rightsquigarrow P :: x{:}A$*; (b)* $\Gamma, u{:}A; \Delta_2 \vdash E \rightsquigarrow Q :: z{:}C$ *with* $Q \overset{\overline{(\nu y)u\langle y\rangle}}{\rightarrow} Q'$*. Then (c)* $\mathsf{cut}^!\ D\ (u.\,E) \equiv\Rightarrow\equiv F$ *for some* F*; (d)* $\Gamma; \Delta \vdash F \rightsquigarrow R :: z{:}C$ *for some* $R \equiv (\nu u)(!u(x).P \mid (\nu y)(P\{y/x\} \mid Q'))$.

Theorem 5.3. *Let* $\Gamma; \Delta \vdash D \rightsquigarrow P :: z{:}A$ *and* $P \rightarrow Q$*. Then there is* E *such that* $D \equiv\Rightarrow\equiv E$ *and* $\Gamma; \Delta \vdash E \rightsquigarrow Q :: z{:}A$

Proof. By induction on the structure of derivation D. The possible cases for D are $D = \mathsf{1L}\ y\ D'$, $D = \mathsf{!L}\ x\ (u.\,D')$, $D = \mathsf{cut}\ D_1\ (x.\,D_2)$, and $D = \mathsf{cut}^!\ D_1\ (x.\,D_2)$, in all other cases $P \not\rightarrow$. Key cases are the cuts, where we rely on a series of reduction lemmas, one for each type C of cut formula, which assign certain proof conversions to process labeled transitions. For example, for $C = C_1 \otimes C_2$, we rely on Lemma 5.1. The case of $\mathsf{cut}^!$, similar to the case $C = !C'$, relies on Lemma 5.2. We show such case in detail. Let $D = \mathsf{cut}^!\ D_1\ (u.\,D_2)$. We have $P \equiv (\nu u)(!u(w).P_1 \mid P_2)$, $\Gamma; \vdash D_1 \rightsquigarrow P_1 :: x{:}C$, and $\Gamma, u : C; \Delta \vdash D_2 \rightsquigarrow P_2 :: z{:}A$ by inversion. Since $P \rightarrow Q$, there two cases: (1) $P_2 \rightarrow Q_2$ and $Q = (\nu u)(!u(w).P_1 \mid Q_2)$, or (2) $P_2 \overset{\alpha}{\rightarrow} Q_2$ where $\alpha = \overline{(\nu y)x\langle y\rangle}$ and $Q = (\nu u)(!u(w).P_1 \mid (\nu y)(P_1\{y/x\} \mid Q_2))$. Case (1): We have $\Gamma, u : C; \Delta \vdash D_2 \rightsquigarrow Q_2 :: z{:}A$ for E' with $D_2 \equiv\Rightarrow\equiv E'$ by i.h. Then $\mathsf{cut}^!\ D_1\ (u.\,D_2) \equiv\Rightarrow\equiv \mathsf{cut}^!\ D_1\ (u.\,E')$ by congruence. Let $E = \mathsf{cut}^!\ D_1\ (u.\,E')$. So $\Gamma; \Delta \vdash E \rightsquigarrow Q :: z{:}A$ by $\mathsf{cut}^!$. Case (2): By Lemma 5.2, $\mathsf{cut}^!\ D_1\ (u.\,D_2) \equiv\Rightarrow\equiv E$ for some E, and $\Gamma; \Delta \vdash E \rightsquigarrow R :: z{:}A$ with $R \equiv Q$. □

Theorems 5.4 and 5.5 state that any proof reduction or conversion also corresponds to either a process equivalence or to a reduction step on the π-calculus.

Theorem 5.4. *Let* $\Gamma; \Delta \vdash D \rightsquigarrow P :: z{:}A$ *and* $D \simeq_s E$*. Then there is* Q *where* $P \simeq_s Q$ *and* $\Gamma; \Delta \vdash E \rightsquigarrow Q :: z{:}A$.

Proof. Following the commuting squares relating $\equiv$, $\rightsquigarrow$ and $\simeq$ in Figures 6, 7 and 8. □

Theorem 5.5. *Let* $\Gamma; \Delta \vdash D \rightsquigarrow P :: z{:}A$ *and* $D \Rightarrow E$*. Then there is* Q *such that* $P \rightarrow Q$ *and* $\Gamma; \Delta \vdash E \rightsquigarrow Q :: z{:}A$.

Proof. Following the commuting squares relating $\Rightarrow$, $\rightsquigarrow$ and $\rightarrow$ in Figure 5. □

Notice that the simulation of π-calculus reductions by proof term conversions provided by Theorem 5.3, and from which subject reduction follows, is very tight indeed, as reduction is simulated up to structural congruence, which is a very fine equivalence on processes. To that end, structural conversions need to be applied symmetrically (as equations), unlike in a standard proof of cut-elimination, where they are usually considered as directed computational steps. Under the assumptions of Theorem 5.3, we can also prove that there is an E such that $D \mapsto\Rightarrow E$ and $\Gamma; \Delta \vdash E \rightsquigarrow R :: z{:}A$, for

$Q \simeq_s R$. Thus, even if one considers the proof conversions as directed reduction rules ($\Rrightarrow$), we still obtain a sound simulation up to typed strong behavioral congruence.

We now state type preservation and progress results for our type system. The subject reduction property (Theorem 5.6) directly follows from Theorem 5.3.

Theorem 5.6 (Subject Reduction). *If $\Gamma;\Delta \vdash P :: z{:}A$ and $P \rightarrow Q$ then $\Gamma;\Delta \vdash Q :: z{:}A$.*

Together with direct consequences of linear typing, Theorem 5.6 ensures session fidelity. Our type discipline also enforces a global progress property. For any P, define

$$\mathit{live}(P) \;\; \mathit{iff} \;\; P \equiv (\nu\overline{n})(\pi.Q \mid R) \quad \mathit{for\ some}\ \pi.Q, R, \overline{n}$$

where $\pi.Q$ is a *non-replicated* guarded process. We first establish the following contextual progress property, from which Theorem 5.8 follows as a corollary.

Lemma 5.7. *Let $\Gamma; \Delta \vdash D \rightsquigarrow P :: z{:}C$. If $\mathit{live}(P)$ then there is Q such that either*

1. *$P \rightarrow Q$, or*
2. *$P \xrightarrow{\alpha} Q$ for α where $s(\alpha) \in (z, \Gamma, \Delta)$. More: if $C = {!A}$ for some A, then $s(\alpha) \neq z$.*

Proof. Induction on derivation D. The key cases are $D = \mathsf{cut}\, D_1\, (y.\, D_2)$ and $D = \mathsf{cut}^!\, D_1(u.\, D_2)$. In the case of cut, we rely on lemmas that characterize the possible actions of a process on name $y{:}A$, depending on type A. These lemmas show that a synchronization between dual actions must occur. For $\mathsf{cut}^!$, an inversion lemma is needed, stating that free names of a non-live process can only be typed by $\mathbf{1}$ or $!A$ types. □

Theorem 5.8 (Progress). *If $\cdot; \cdot \vdash D \rightsquigarrow P :: x{:}\mathbf{1}$ and $\mathit{live}(P)$ then exists Q st. $P \rightarrow Q$.*

Proof. By Lemma 5.7 and the fact that P cannot perform any action α with subject $s(\alpha) = x$ since $x{:}\mathbf{1}$ (by the action shape characterization lemmas). □

6 Discussion and Further Examples

We further compare our linear type system for (finite) session types with more familiar session type systems [21,18,13]. An immediate observation is that in our case types are freely generated, while traditionally there is a stratification of types in "session" and "standard types" (the later corresponding to our $!A$ types, typing session initiation channels). In our interpretation, a session may either terminate ($\mathbf{1}$), or become a replicated server ($!A$), which is more general and uniform, and a natural consequence of the logical interpretation. Concerning parallel composition, usually two rules can be found, one corresponding to the cancellation of two dual session endpoints (a name restriction rule), and another corresponding to independent parallel composition, also present in most linear type systems for mobile processes. In our case, cut combines both principles, and the following rule is derivable:

$$\frac{\Gamma; \Delta \vdash P :: -{:}\mathbf{1} \quad \Gamma; \Delta' \vdash Q :: T}{\Gamma; \Delta, \Delta' \vdash P \mid Q :: T} \ (\mathsf{comp})$$

A consequence of the logical composition rules cut and $\mathsf{cut}^!$ is that typing intrinsically enforces global progress, unlike with traditional session type systems [18,13], which do not ensure progress in the presence of multiple open sessions, as we do here. Techniques to ensure progress in sessions, but building on extraneous devices such as well-founded orderings on events, have been proposed [20,12]. It would be interesting to further compare the various approaches, as far as process typability is concerned.

Channel "polarities" are captured in our system by the left-right distinction of sequents, rather than by annotations on channels (cf. x^+, x^-). Session and linear type systems [21,18,13] also include a typing rule for output of the form

$$\frac{\Gamma; \Delta \vdash P :: x{:}C}{\Gamma; \Delta, y{:}A \vdash x\langle y\rangle.P :: x{:}A \otimes C}$$

In our case, an analogous rule may be derived by $\otimes$R and the copycat construction, where a "proxy" for the free name y, bidirectionally copying behavior A, is linked to z.

$$\frac{\Gamma; \Delta \vdash P :: x{:}C}{\Gamma; \Delta, y{:}A \vdash (\nu z)x\langle z\rangle.(id_A(y,z) \mid P) :: x{:}A \otimes C}$$

The copycat $id_A(y,z)$ plays the role of the "link" processes of [23,7]. Notice that in our case the definition of the "link" is obtained for free by the interpretation of identity axioms (Proposition 4.2). The two processes can be shown to be behaviorally equivalent, under an adequate notion of observational equivalence, as in [7].

We now elaborate on the example of Section 3, in order to illustrate sharing and session initiation. Consider now a different client, that picks the "buy" rather than the "quote" operation, and the corresponding composed system.

$$\begin{aligned} \mathit{BClntBody}_s &\triangleq s.\mathtt{inl}; (\boldsymbol{\nu}\mathit{cof})s\langle \mathit{cof}\rangle.(\boldsymbol{\nu}\mathit{pin})s\langle \mathit{pin}\rangle.s(\mathit{rc})\mathbf{0} \\ \mathit{BSimple} &\triangleq (\boldsymbol{\nu}s)(\mathit{SrvBody}_s \mid \mathit{BClntBody}_s) \end{aligned}$$

We have the typings $\cdot; s{:}\mathit{ServerProto} \vdash \mathit{BClntBody}_s :: -{:}\mathbf{1}$ and $\cdot;\cdot \vdash \mathit{BSimple} :: -{:}\mathbf{1}$.

In these examples, there is a single installed pair client-server, where the session is already initiated, and only known to the two partners. To illustrate sharing, we now consider a replicated server. Such a replicated server is able to spawn a fresh session instance for each initial invocation, each one conforming to the general behavior specified by *ServerProto*, and can be typed by !*ServerProto*. Correspondingly, clients must initially invoke the replicated server to instantiate a new session (cf. the Tcopy rule).

$$\begin{aligned} \mathit{QClient} &\triangleq (\boldsymbol{\nu}s)c\langle s\rangle.QClntBody_s & \mathit{BClient} &\triangleq (\boldsymbol{\nu}s)c\langle s\rangle.BClntBody_s \\ \mathit{Server} &\triangleq\, !c(s).SrvBody_s & \mathit{SharSys} &\triangleq (\boldsymbol{\nu}c)(\mathit{Server} \mid \mathit{BClient} \mid \mathit{QClient}) \end{aligned}$$

For the shared server, by T!R, we type $\cdot;\cdot \vdash \mathit{Server} :: c{:}!\mathit{ServerProto}$. We also have, for the clients, by Tcopy the typings $c{:}\mathit{ServerProto}\ ;\ \cdot \vdash \mathit{BClient} :: -{:}\mathbf{1}$ and $c{:}\mathit{ServerProto}\ ;\ \cdot \vdash \mathit{QClient} :: -{:}\mathbf{1}$. By (comp), T!L, and Tcut we obtain the intended typing for the whole system: $\cdot;\cdot \vdash \mathit{SharSys} :: - : \mathbf{1}$. Notice how the session instantiation protocol is naturally explained by the logical interpretation of the ! operator.

7 Related Work and Conclusions

We have established a tight correspondence between a session-based type discipline for the π-calculus and intuitionistic linear logic: typing rules correspond to dual intuitionistic linear sequent calculus proof rules, moreover process reduction may be simulated in a type preserving way by proof conversions and reductions, and *vice versa*. As a result, we obtain the subject reduction property, from which session fidelity follows. Our basic typing discipline intrinsically ensures global progress, beyond the restricted "progress on a single session" property obtained in pure session type systems.

Other works have investigated π-calculus models of linear logic proofs. Bellin and Scott [5] establish a mapping from linear logic proofs to a variant of the π-calculus and some connections between proof reduction and π-calculus reduction. However, this mapping results in complex encodings, so that their system could hardly be considered a type assignment system for processes, which has been achieved in this work. Moreover, no relation between behavioral descriptions and logical propositions was identified, as put by the authors: "[our encodings] have less to do with logic than one might think, they are essentially only about the abstract pluggings in proof structures". A realizability interpretation for a linear logic augmented with temporal modalities (cf. Hennessy-Milner) was proposed in [4], also based on a π-calculus variant. A recent related development is [17], where a correspondence between (independently formulated) proof nets and an IO-typed π-calculus is established. In our case, the type system and the logic proof system are exactly the same, and we reveal a direct connection between pure linear logic propositions and behavioral types on π-calculus, that covers all (both shared and linear) features of finite session types. A development of session types as linear process types (in the sense of [21]) is presented in [15], where linearity and sharing are expressed by special annotations, unrelated to a linear logic interpretation.

We have also analyzed the relation between our type discipline and (finite, deadlock-free) session types. It is important to notice that our interpretation does not require locality for session (linear) channels (under which only the output capability of names could be transmitted), which seems required in other works on linearity for π-calculi (*e.g.*, [26]). On the other hand, our intuitionistic discipline enforces locality of shared channels, which, quite interestingly, seems to be the sensible choice for distributed implementations of sessions. Interesting related topics would be the accommodation of recursive types, logical relations [8], and the characterization of observational equivalences under our typing discipline. In particular, we expect that all conversions (including commuting conversions) between DILL derivations correspond to observational equivalences on our typed π-calculus.

One important motivation for choosing a purely logical approach to typing is that it often suggests uniform and expressive generalizations. In ongoing work, we have also established an explicit relationship between session-based concurrency and functional computation where in both cases determinacy (no races) and progress (deadlock-freedom) are expected features. In particular, we have been investigating new encodings of λ-calculi into the π-calculus that arise from translations from DILL natural deduction into sequent calculus. We also believe that dependent generalizations of our system of simple linear types, perhaps along the lines of LLF [9] or CLF [25], may be able to capture many additional properties of communication behavior in a purely logical manner. Already, some systems of session types have dependent character, such as [6] that, among other properties, integrates correspondence assertions into session types.

Acknowledgments. To FCT/MCTES (INTERFACES NGN44), and the ICTI at Carnegie-Mellon. Thanks also to Bernardo Toninho, Nobuko Yoshida, and Andre Platzer.

References

1. Abramsky, S.: Computational Interpretations of Linear Logic. TCS 111(1&2) (1993)
2. Andreoli, J.-M.: Logic Programming with Focusing Proofs in Linear Logic. Journal of Logic and Computation 2(3), 197–347 (1992)

3. Barber, A., Plotkin, G.: Dual Intuitionistic Linear Logic. Technical Report LFCS-96-347, Univ. of Edinburgh (1997)
4. Beffara, E.: A Concurrent Model for Linear Logic. ENTCS 155, 147–168 (2006)
5. Bellin, G., Scott, P.: On the π-Calculus and Linear Logic. TCS 135, 11–65 (1994)
6. Bonelli, E., Compagnoni, A., Gunter, E.L.: Correspondence Assertions for Process Synchronization in Concurrent Communications. J. of Func. Prog. 15(2), 219–247 (2005)
7. Boreale, M.: On the Expressiveness of Internal Mobility in Name-Passing Calculi. Theoretical Computer Science 195(2), 205–226 (1998)
8. Caires, L.: Logical semantics of types for concurrency. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 16–35. Springer, Heidelberg (2007)
9. Cervesato, I., Pfenning, F.: A Linear Logical Framework. Inf. & Comput. 179(1) (2002)
10. Chang, B.-Y.E., Chaudhuri, K., Pfenning, F.: A Judgmental Analysis of Linear Logic. Technical Report CMU-CS-03-131R, Carnegie Mellon University (2003)
11. Dezani-Ciancaglini, M., de' Liguoro, U.: Sessions and Session Types: an Overview. In: 6th Intl. Workshop on Web Services and Formal Methods WS-FM 2009. LNCS. Springer, Heidelberg (2010)
12. Dezani-Ciancaglini, M., de' Liguoro, U., Yoshida, N.: On Progress for Structured Communications. In: Barthe, G., Fournet, C. (eds.) TGC 2007 and FODO 2008. LNCS, vol. 4912, pp. 257–275. Springer, Heidelberg (2008)
13. Gay, S., Hole, M.: Subtyping for Session Types in the Pi Calculus. Acta Informatica 42(2-3), 191–225 (2005)
14. Girard, J.-Y., Lafont, Y.: Linear Logic and Lazy Computation. In: Ehrig, H., Kowalski, R.A., Levi, G., Montanari, U. (eds.) TAPSOFT 1987 and CFLP 1987. LNCS, vol. 250, pp. 52–66. Springer, Heidelberg (1987)
15. Giunti, M., Vasconcelos, V.T.: A Linear Account of Session Types in the Pi-Calculus. In: Gastin, P., Laroussinie, F. (eds.) 21st International Conference on Concurrency Theory, Concur 2010. Springer, Heidelberg (2010)
16. Honda, K.: Types for Dyadic Interaction. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 509–523. Springer, Heidelberg (1993)
17. Honda, K., Laurent, O.: An Exact Correspondence between a Typed pi-calculus and Polarised Proof-Nets. Theoretical Computer Science (to appear, 2010)
18. Honda, K., Vasconcelos, V.T., Kubo, M.: Language Primitives and Type Discipline for Structured Communication-Based Programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998)
19. Hyland, J.M.E., Luke Ong, C.-H.: Pi-Calculus, Dialogue Games and PCF. In: WG2.8 Conference on Functional Programming Languages, pp. 96–107 (1995)
20. Kobayashi, N.: A Partially Deadlock-Free Typed Process Calculus. ACM Tr. Progr. Lang. Sys. 20(2), 436–482 (1998)
21. Kobayashi, N., Pierce, B.C., Turner, D.N.: Linearity and the Pi-Calculus. In: 23rd Symp. on Principles of Programming Languages, POPL 1996, pp. 358–371. ACM, New York (1996)
22. Milner, R.: Functions as processes. Math. Struc. in Computer Sciences 2(2), 119–141 (1992)
23. Sangiorgi, D.: Pi-Calculus, Internal Mobility, and Agent Passing Calculi. Theoretical Computer Science 167(1&2), 235–274 (1996)
24. Sangiorgi, D., Walker, D.: The π-calculus: A Theory of Mobile Processes. CUP, Cambridge (2001)
25. Watkins, K., Cervesato, I., Pfenning, F., Walker, D.: Specifying properties of concurrent computations in CLF. In: Schürmann, C. (ed.) 4th Intl. Workshop on Logical Frameworks and Meta-Languages (LFM 2004), Cork, Ireland, July 2004. ENTCS, vol. 199 (2004)
26. Yoshida, N., Honda, K., Berger, M.: Linearity and Bisimulation. J. Logic and Algebraic Programming 72(2), 207–238 (2007)

Session Types for Access and Information Flow Control*

Sara Capecchi[1], Ilaria Castellani[2],
Mariangiola Dezani-Ciancaglini[1], and Tamara Rezk[2]

[1] Dipartimento di Informatica, Università di Torino, corso Svizzera 185, 10149 Torino, Italy
[2] INRIA, 2004 route des Lucioles, 06902 Sophia Antipolis, France

Abstract. We consider a calculus for multiparty sessions with delegation, enriched with security levels for session participants and data. We propose a type system that guarantees both session safety and a form of access control. Moreover, this type system ensures secure information flow, including controlled forms of declassification. In particular, the type system prevents leaks that could result from an unrestricted use of the control constructs of the calculus, such as session opening, selection, branching and delegation. We illustrate the use of our type system with a number of examples, which reveal an interesting interplay between the constraints used in security type systems and those used in session types to ensure properties like communication safety and session fidelity.

Keywords: concurrency, communication-centred computing, session types, access control, secure information flow.

1 Introduction

With the advent of web technologies and the proliferation of programmable and interconnectable devices, we are faced today with a powerful and heterogeneous computing environment. This environment is inherently parallel and distributed and, unlike previous computing environments, it heavily relies on communication. It therefore calls for a new programming paradigm which is sometimes called *communication-centred*. Moreover, since computations take place concurrently in all kinds of different devices, controlled by parties which possibly do not trust each other, security properties such as the confidentiality and integrity of data become of crucial importance. The issue is then to develop models, as well as programming abstractions and methodologies, to be able to exploit the rich potential of this new computing environment, while making sure that we can harness its complexity and get around its security vulnerabilities. To this end, calculi and languages for communication-centred programming have to be *security-minded* from their very conception, and make use of specifications not only for data structures, but also for communication interfaces and for security properties.

The aim of this paper is to investigate type systems for safe and secure sessions. A *session* is an abstraction for various forms of "structured communication" that may occur in a parallel and distributed computing environment. Examples of sessions are

* Work partially funded by the INRIA Sophia Antipolis COLOR project MATYSS, by the ANR-SETI-06-010 and ANR-08-EMER-010 grants, and by the MIUR Projects DISCO and IPODS.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 237–252, 2010.

a client-service negotiation, a financial transaction, or a multiparty interaction among different services within a web application.

Language-based support for sessions has now become the subject of active research. Primitives for enabling programmers to code sessions in a flexible way, as well as type systems ensuring the compliance of programs to session specifications (session types), have been studied in a variety of calculi and languages in the last decade. *Session types* were originally introduced in a variant of the pi-calculus [20]. We refer to [7] for a survey on the session type literature. The key properties ensured by session types are *communication safety*, namely the consistency of the communication patterns exhibited by the partners (implying the absence of communication errors), and *session fidelity*, ensuring that channels which carry messages of different types do it in a specific order.

Enforcement of security properties via session types has been studied in [3,15]. These papers propose a compiler which, given a multiparty session description, implements cryptographic protocols that guarantee *session execution integrity*. The question of ensuring *access control* in binary sessions has been recently addressed in [13] for the Calculus of Services with Pipelines and Sessions of [4], where delegation is absent. On the other hand, the property of *secure information flow* has not been investigated within session calculi so far. This property, first studied in the early eighties [9], has regained interest in the last decade, due to the evolution of the computing environment. It has now been thoroughly studied for both programming languages (cf [16] for a review) and process calculi [8,10,12].

In this paper, we address the question of incorporating mandatory access control and secure information flow within session types. We consider a calculus for multiparty sessions with delegation, enriched with security levels for both session participants and data, and providing a form of declassification for data [18], as required by most practical applications. We propose a type system that ensures access control, namely that each participant receives data of security level less than or equal to its own. For instance, in a well-typed session involving a Customer, a Seller and a Bank, the secret credit card number of the Customer will be communicated to the Bank, but not to the Seller. Moreover, our type system prevents insecure flows that could occur via the specific constructs of the language, such as session opening, selection, branching and delegation. Finally, we show that it allows controlled forms of declassification, namely those permitted by the access control policy. Our work reveals an interesting interplay between the constraints of security type systems and those used in session types to ensure properties like communication safety and session fidelity.

The rest of the paper is organised as follows. In Section 2 we motivate our access control and declassification policies with an example. Section 3 introduces the syntax and semantics of our calculus. In Section 4 we define the secure information flow property. In Section 5 we illustrate this property by means of examples. Section 6 presents our type system for safe and secure sessions and theorems establishing its soundness. Section 7 concludes with a discussion on future work. The reader is referred to the full paper (available at http://hal.inria.fr/INRIA) for complete definitions and proofs.

2 A First Example on Access Control and Declassification

In this section we illustrate by an example the basic features of our typed calculus, as well as our access control policy and its use for declassification. The question of secure information flow will only be marginal here. It will be discussed in Sections 4 and 5.

A client C sends the title of a book to a bookseller S. Then S *delegates* to a bank B both the reception of the credit card number of C and the control of its validity. This delegation is crucial for assuring the secrecy of the credit card number, which should be read by B but not by S. Then B notifies S about the result of the control: for this a *declassification* is needed. Finally, if the credit card is valid, C receives a delivery date from S, otherwise the deal falls through. More precisely, the protocol is as follows:

1. C opens a connection with S and sends a title to S;
2. S opens a connection with B and *delegates* to B part of his conversation with C;
3. C sends his secret credit card number *apparently* to the untrusted party S but *really* - thanks to delegation - to the trusted party B;
4. B delegates back to S the conversation with C;
5. B selects the answer ok or ko for S depending on the validity of the credit card, thus performing a declassification;
6. S sends to C either ok and a date, or just ko, depending on the label ok or ko chosen by B.

In our calculus, which is an enrichment with security levels of the calculus in [2], this scenario may be described as the parallel composition of the following processes, where security levels appear as superscripts on both data and operators (here we omit unnecessary levels on operators and use $\perp$ to mean "public" and $\top$ to mean "secret"):

$$\mathtt{I} = \bar{a}[2] \mid \bar{b}[2]$$

$$\mathtt{C} = a[1](\alpha_1).\alpha_1!\langle 2, \mathsf{Title}^{\perp}\rangle.\alpha_1!^{\perp}\langle 2, \mathsf{CreditCard}^{\top}\rangle.\alpha_1\&(2, \{\mathsf{ok} : \alpha_1?(2, \mathit{date}^{\perp}).\mathbf{0}, \mathsf{ko} : \mathbf{0}\})$$

$$\begin{aligned}\mathtt{S} = {} & a[2](\alpha_2).\alpha_2?(1, x^{\perp}).b[2](\beta_2).\beta_2!\langle\langle 1, \alpha_2\rangle\rangle.\beta_2?((1, \zeta)).\\ & \beta_2\&(1, \{\mathsf{ok} : \zeta \oplus \langle 1, \mathsf{ok}\rangle.\zeta!\langle 1, \mathsf{Date}^{\perp}\rangle.\mathbf{0}, \mathsf{ko} : \zeta \oplus \langle 1, \mathsf{ko}\rangle.\mathbf{0}\})\end{aligned}$$

$$\begin{aligned}\mathtt{B} = {} & b[1](\beta_1).\beta_1?((2, \zeta)).\zeta?^{\top}(2, cc^{\perp}).\beta_1!\langle\langle \zeta, 2\rangle\rangle.\\ & \mathsf{if}\ \mathit{valid}(cc^{\perp})\ \mathsf{then}\ \beta_1 \oplus \langle 2, \mathsf{ok}\rangle.\mathbf{0}\ \mathsf{else}\ \beta_1 \oplus \langle 2, \mathsf{ko}\rangle.\mathbf{0}\end{aligned}$$

A session is a particular activation of a service, involving a number of parties with predefined roles. Here processes C and S communicate by opening a session on service *a*, while processes S and B communicate by opening a session on service *b*. The initiators $\bar{a}[2]$ and $\bar{b}[2]$ specify the number of participants of each service. We associate integers with participants in services: here C=1, S=2 in service *a* and B=1, S=2 in service *b*.

In process C, the prefix $a[1](\alpha_1)$ means that C wants to act as participant 1 in service *a* using channel α_1, matching channel α_2 of participant 2, who is S. When the session is established, C sends to S a title of level $\perp$ and a credit card number of level $\top$, indicating (by the superscript $\perp$ on the output operator) that the credit card number may be declassified to $\perp$. Then he waits for either ok, followed by a date, or ko.

indent Process S receives a value in service *a* and then enters service *b* as participant 2. Here the output $\beta_2!\langle\langle 1, \alpha_2\rangle\rangle$ sends channel α_2 to the participant 1 of *b*, who is B, thus delegating to B the use of α_2. Then S waits for a channel ζ from B. Henceforth,

Table 1. Global types of the B, C, S example

1. $\mathtt{C} \to \mathtt{S} : \left\langle \mathsf{String}^{\perp} \right\rangle$	
2. $\mathtt{S} \Rsh \delta$	$\mathtt{S} \to \mathtt{B} : \langle T \rangle$
3. $\mathtt{C} \to \mathtt{S} : \left\langle \mathsf{Number}^{\top\downarrow\perp} \right\rangle$	
4. $\mathtt{S} \Rsh \delta$	$\mathtt{B} \to \mathtt{S} : \langle T' \rangle$
5.	$\mathtt{B} \to \mathtt{S} : \{\mathsf{ok} : \mathsf{end}, \mathsf{ko} : \mathsf{end}\}$
6. $\mathtt{S} \to \mathtt{C} : \{\mathsf{ok} : \mathtt{S} \to \mathtt{C} : \left\langle \mathsf{String}^{\perp} \right\rangle ; \mathsf{end}, \mathsf{ko} : \mathsf{end}\}$	

S communicates using both channels β_2 and ζ: on channel β_2 he waits for one of the labels ok or ko, which he then forwards to C on ζ, sending also a date if the label is ok.

Forgetting session opening and abstracting from values to types, we may represent the whole communication protocol by the *global types* of Table 1 (where we use B, C, S instead of 1, 2), where the left-hand side and right-hand side describe services a and b, respectively. Line 1 says that C sends a String of level $\perp$ to S. In line 2, $\mathtt{S}\Rsh\delta$ means that the channel from S to C is delegated: this delegation is realised by the transmission of the channel (with type T) from S to B, as shown on the right-hand side. Line 3 says that C sends a Number of level $\top$ to S, allowing him to declassify it to $\perp$. Notice that due to the previous delegation the Number is received by B and not by S. Line 4 describes a delegation which is the inverse of that in Line 2: here the (behavioural) type of the channel has changed, since the channel has already been used to receive the Number. Line 5 says that B sends to S one of the labels ok or ko. Finally, line 6 says that S sends to C either the label ok followed by a String of level $\perp$, or the label ko. Since B's choice of the label ok or ko depends on a test on the Number, it is crucial that Number be previously declassified to $\perp$, otherwise the reception of a String of level $\perp$ by C would depend on a value of level $\top$ (this is where secure information flow comes into play).

Type T represents the conversation between C and S after the first communication, seen from the viewpoint of S. Convening that $?(-), !\langle - \rangle$ represent input and output in types, that ";" stands for sequencing and that $\oplus\langle - \{-\}\rangle$ represents the choice of sending one among different labels, it is easy to see that the *session type* T is:

$$?\left(\mathtt{C}, \mathsf{Number}^{\top\downarrow\perp}\right); \oplus\left\langle \mathtt{C}, \{\mathsf{ok} : !\left\langle \mathtt{C}, \mathsf{String}^{\perp} \right\rangle ; \mathsf{end}, \mathsf{ko} : \mathsf{end}\} \right\rangle$$

where the communication partner of S (namely C) is explicitly mentioned. The session type T' is the rest of type T after the first communication has been done:

$$\oplus\left\langle \mathtt{C}, \{\mathsf{ok} : !\left\langle \mathtt{C}, \mathsf{String}^{\perp} \right\rangle ; \mathsf{end}, \mathsf{ko} : \mathsf{end}\} \right\rangle$$

To formalise access control, we will give security levels to service participants, and require that a participant of a given level does not receive data of higher or incomparable level. Since the only secret data in our example is CreditCard, it is natural to associate $\perp$ with S in both services a and b, and $\top$ with B in service b. Notice that C may indifferently have level $\top$ or $\perp$, since it only sends, but does not receive, the high data CreditCard.

3 Syntax and Semantics

Our calculus for multiparty asynchronous sessions is essentially the same as that considered in [2], with the addition of runtime configurations and security levels.

Table 2. Syntax of expressions, processes, queues, and configurations

$P ::= \bar{u}[n]$	n-ary session initiator		
$\mid\ u[\mathrm{p}](\alpha).P$	p-th session participant	$c ::= \alpha \mid s[\mathrm{p}]$	Channel
$\mid\ c!^{\ell}\langle \Pi, e\rangle.P$	Value sending	$e ::= v^{\ell} \mid x^{\ell}$	
$\mid\ c?^{\ell}(\mathrm{p}, x^{\ell'}).P$	Value receiving	$\mid\ e$ and $e' \mid$ not $e \ldots$	Expression
$\mid\ c!^{\ell}\langle\langle \mathrm{q}, c'\rangle\rangle.P$	Delegation sending		
$\mid\ c?^{\ell}((\mathrm{p}, \alpha)).P$	Delegation reception	$D ::= X(x^{\ell}, \alpha) = P$	Declaration
$\mid\ c \oplus^{\ell} \langle \Pi, \lambda\rangle.P$	Selection	$\Pi ::= \{\mathrm{p}\} \mid \Pi \cup \{\mathrm{p}\}$	Set of participants
$\mid\ c\&^{\ell}(\mathrm{p}, \{\lambda_i : P_i\}_{i\in I})$	Branching		
$\mid$ if e then P else Q	Conditional	$\vartheta ::= v^{\ell\downarrow\ell'} \mid s[\mathrm{p}]^{\ell} \mid \lambda^{\ell}$	Message content
$\mid\ P \mid Q$	Parallel		
$\mid\ \mathbf{0}$	Inaction	$m ::= (\mathrm{p}, \Pi, \vartheta)$	Message in transit
$\mid\ (\nu a^{\ell})P$	Name hiding		
$\mid$ def D in P	Recursion	$h ::= m \cdot h \mid \varepsilon$	Queue
$\mid\ X\langle e, c\rangle$	Process call		
		$H ::= H \cup \{s : h\} \mid \emptyset$	**Q**-set
$u ::= x^{\ell} \mid a^{\ell}$	Identifier	$r ::= a^{\ell} \mid s$	Service/Session Name
$v ::= a \mid$ true $\mid$ false $\mid \ldots$	Value		

Syntax. Let $(\mathscr{S}, \sqsubseteq)$ be a finite lattice of *security levels*, ranged over by ℓ, ℓ'. We denote by $\sqcup$ and $\sqcap$ the join and meet operations on the lattice, and by $\bot$ and $\top$ its minimal and maximal elements.

We assume the following sets: *service names*, ranged over by $a, b, \ldots$ each of which has an *arity* $n \geq 2$ (its number of participants) and a security level ℓ, *value variables*, ranged over by $x, y, \ldots$, all decorated with security levels, *identifiers*, i.e., service names and value variables, ranged over by $u, w, \ldots$, all decorated with security levels, *channel variables*, ranged over by $\alpha, \beta, \ldots$, *labels*, ranged over by $\lambda, \lambda', \ldots$ (acting like labels in labelled records). *Values* v are either service names or basic values (boolean values, integers, etc.). When treated as an expression, a value is decorated with a security level ℓ; when used in a message, it is decorated with a declassified level of the form $\ell \downarrow \ell'$, where $\ell' \leq \ell$ (in case $\ell' = \ell$, we will write simply ℓ instead of $\ell \downarrow \ell$).

Sessions, the central abstraction of our calculus, are denoted with $s, s' \ldots$. A session represents a particular instance or activation of a service. Hence sessions only appear at runtime. We use p, q,... to denote the *participants of a session. In an n-ary session (a session corresponding to an n-ary service)* p, q *are assumed to range over the natural numbers* $1, \ldots, n$. We denote by Π a non empty set of participants. Each session s has an associated set of *channels with role* $s[\mathrm{p}]$, one for each participant. Channel $s[\mathrm{p}]$ is the private channel through which participant p communicates with the other participants in the session s. A new session s on an n-ary service a^{ℓ} is opened when the *initiator* $\bar{a}^{\ell}[n]$ of the service synchronises with n processes of the form $a^{\ell}[\mathrm{p}](\alpha).P$. We use c to range over channel variables and channels with roles. Finally, we assume a set of *process variables* $X, Y, \ldots$, in order to define recursive behaviours.

The set of *expressions*, ranged over by $e, e', \ldots$, and the set of *processes*, ranged over by $P, Q \ldots$, are given by the grammar in Table 2, where syntax occurring only at runtime appears shaded. The primitives are decorated with security levels. When there is no risk of confusion we will omit the set delimiters $\{,\}$.

As in [11], in order to model TCP-like asynchronous communications (with non-blocking send but message order preservation between a given pair of participants), we use *queues of messages*, denoted by h; an element of h may be a value message $(\mathtt{p},\Pi,v^{\ell\downarrow\ell'})$, indicating that the value v^{ℓ} is sent by participant $\mathtt{p}$ to all participants in Π, with the right of declassifying it from ℓ to ℓ'; a channel message $(\mathtt{p},\mathtt{q},s[\mathtt{p}']^{\ell})$, indicating that $\mathtt{p}$ delegates to $\mathtt{q}$ the role of $\mathtt{p}'$ with level ℓ in the session s; and a label message $(\mathtt{p},\Pi,\lambda^{\ell})$, indicating that $\mathtt{p}$ selects the process with label λ among the processes offered by the set of participants Π. The empty queue is denoted by ε, and the concatenation of a new message m to a queue h by $h\cdot m$. Conversely, $m\cdot h$ means that m is the head of the queue. Since there may be nested and parallel sessions, we distinguish their queues by naming them. We denote by $s:h$ the *named queue* h associated with session s. We use H,K to range over sets of named queues, also called **Q**-sets.

Operational Semantics. The operational semantics is defined on configurations. A *configuration* is a pair $C = < P\ ,\ H >$ of a process P and a **Q**-set H, possibly restricted with respect to service and session names, or a parallel composition of configurations, denoted by $C \| C$. In a configuration $(\nu s) < P\ ,\ H >$, all occurrences of $s[\mathtt{p}]$ in P and H and of s in H are bound. By abuse of notation we will often write P instead of $< P\ ,\ \emptyset >$.

We use a *structural equivalence* $\equiv$ [14] for processes, queues and configurations. Modulo $\equiv$, each configuration has the form $(\nu\tilde{r}) < P\ ,\ H >$, where $(\nu\tilde{r})C$ stands for $(\nu r_1)\cdots(\nu r_k)C$, if $\tilde{r} = r_1\cdots r_k$. In $(\nu a^{\ell})C$, we assume that α-conversion on the name a^{ℓ} preserves the level ℓ. Among the rules for queues, we have one for commuting independent messages and another one for splitting a message for multiple recipients.

The transitions for configurations have the form $C \longrightarrow C'$. They are derived using the reduction rules in Table 3. Rule [Link] describes the initiation of a new session among n processes, corresponding to an activation of the service a^{ℓ} of arity n. After the connection, the participants share a private session name s and the corresponding queue, initialised to $s:\varepsilon$. The variable $\alpha_{\mathtt{p}}$ in each participant $P_{\mathtt{p}}$ is replaced by the corresponding channel with role $s[\mathtt{p}]$. The output rules [Send], [DelSend] and [Label] push values, channels and labels, respectively, into the queue $s:h$. In rule [Send], $e\downarrow v^{\ell}$ denotes the evaluation of the expression e to the value v^{ℓ}, where ℓ is the join of the security levels of the variables and values occurring in e. The superscript ℓ' on the output sign indicates that v^{ℓ} can be declassified to level ℓ', when received by an input process $s[\mathtt{q}]?^{\ell}(\mathtt{p},x^{\ell'}).P$. This is why the value is recorded with both levels in the queue. The rules [Rec], [DelRec] and [Branch] perform the corresponding complementary operations. As usual, we will use $\longrightarrow^{*}$ for the reflexive and transitive closure of $\longrightarrow$.

4 Information Flow Security in Sessions

We turn now to the question of ensuring *secure information flow* [6] within sessions. We shall be interested in the property of *noninterference* (NI) [9], combined with a limited form of *declassification* [1], which may only take place during a value communication. The property of NI requires that there is no flow of information from objects of a given level to objects of lower or incomparable level [21,19,16]. To set the stage for our information flow analysis, the first questions to ask are:

1. Which objects of the calculus should carry security levels?
2. Which information leaks can occur and how can they be detected?

As concerns objects, we shall see that besides values, also labels, delegated channels and services will need security levels. Since this question requires some discussion, which is best understood through examples, we defer it to the next section, just assuming here as a fact that queue messages have the form $(\mathtt{p}, \Pi, \vartheta)$, where ϑ may be $v^{\ell\downarrow\ell'}$, λ^{ℓ} or $s[\mathtt{p}]^{\ell}$. In the rest of this section, we will focus on the observation model, which will be based on bisimulation, as is now standard for concurrent processes [19,17].

We assume that the observer can see the content of messages in session queues. To fix ideas, one may view the observer as a kind of buffer through which messages may transit while reaching or leaving a session queue. We do not want to go as far as allowing an observer to take part in a session, since that could affect the behaviour of other processes. In other words, we assume a passive observer rather than an active one.

What matters for security is observation relative to a given set of levels. Given a downward-closed subset $\mathscr{L}$ of $\mathscr{S}$, a $\mathscr{L}$-observer will only be able to see messages whose level belongs to $\mathscr{L}$. A notion of $\mathscr{L}$-equality $=_{\mathscr{L}}$ on **Q**-sets is then introduced, representing indistinguishability of **Q**-sets by a $\mathscr{L}$-observer. Based on $=_{\mathscr{L}}$, a notion of $\mathscr{L}$-bisimulation $\simeq_{\mathscr{L}}$ will formalise indistinguishability of processes by a $\mathscr{L}$-observer.

Formally, a queue $s : h$ is $\mathscr{L}$-observable if it contains some message with a level in $\mathscr{L}$. Then two **Q**-sets are $\mathscr{L}$*-equal* if their $\mathscr{L}$-observable queues have the same names and contain the same messages with a level in $\mathscr{L}$. This equality is based on a $\mathscr{L}$-projection operation on **Q**-sets, which discards all messages whose level is not in $\mathscr{L}$.

Definition 1. *Let the functions $lev_{\uparrow}$ and $lev_{\downarrow}$ be defined by:* $lev_{\uparrow}(v^{\ell\downarrow\ell'}) = \ell$, $lev_{\downarrow}(v^{\ell\downarrow\ell'}) = \ell'$, *and* $lev_{\uparrow}(s[\mathtt{p}]^{\ell}) = lev_{\uparrow}(\lambda^{\ell}) = \ell = lev_{\downarrow}(s[\mathtt{p}]^{\ell}) = lev_{\downarrow}(\lambda^{\ell})$.

Definition 2. *The projection operation $\Downarrow \mathscr{L}$ is defined inductively on messages, queues and* **Q***-sets as follows:*

Table 3. Reduction rules (excerpt)

$a^{\ell}[1](\alpha_1).P_1 \mid \ldots \mid a^{\ell}[n](\alpha_n).P_n \mid \bar{a}^{\ell}[n] \longrightarrow (\nu s) < P_1\{s[1]/\alpha_1\} \mid \ldots \mid P_n\{s[n]/\alpha_n\} \, , \, s : \varepsilon >$	[Link]
$< s[\mathtt{p}]!^{\ell'}\langle \Pi, e\rangle.P \, , \, s : h > \longrightarrow < P \, , \, s : h \cdot (\mathtt{p}, \Pi, v^{\ell\downarrow\ell'}) > \qquad (e{\downarrow}v^{\ell})$	[Send]
$< s[\mathtt{q}]?^{\ell}(\mathtt{p}, x^{\ell'}).P \, , \, s : (\mathtt{p}, \mathtt{q}, v^{\ell\downarrow\ell'}) \cdot h > \longrightarrow < P\{v^{\ell'}/x^{\ell'}\} \, , \, s : h >$	[Rec]
$< s[\mathtt{p}]!^{\ell}\langle\langle \mathtt{q}, s'[\mathtt{p}']\rangle\rangle.P \, , \, s : h > \longrightarrow < P \, , \, s : h \cdot (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']^{\ell}) >$	[DelSend]
$< s[\mathtt{q}]?^{\ell}((\mathtt{p}, \alpha)).P \, , \, s : (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']^{\ell}) \cdot h > \longrightarrow < P\{s'[\mathtt{p}']/\alpha\} \, , \, s : h >$	[DelRec]
$< s[\mathtt{p}] \oplus^{\ell} \langle \Pi, \lambda\rangle.P \, , \, s : h > \longrightarrow < P \, , \, s : h \cdot (\mathtt{p}, \Pi, \lambda^{\ell}) >$	[Label]
$< s[\mathtt{q}]\&^{\ell}(\mathtt{p}, \{\lambda_i : P_i\}_{i \in I}) \, , \, s : (\mathtt{p}, \mathtt{q}, \lambda^{\ell}_{i_0}) \cdot h > \longrightarrow < P_{i_0} \, , \, s : h > \quad (i_0 \in I)$	[Branch]
$C \longrightarrow (\nu\tilde{s})C' \quad \Rightarrow \quad (\nu\tilde{r})(C \parallel C'') \longrightarrow (\nu\tilde{r})(\nu\tilde{s})(C' \parallel C'')$	[ScopC]

$$(\mathrm{p},\Pi,\vartheta)\Downarrow\mathscr{L}=\begin{cases}(\mathrm{p},\Pi,\vartheta) & \textit{if } lev_{\downarrow}(\vartheta)\in\mathscr{L},\\ \varepsilon & \textit{otherwise.}\end{cases}$$

$$\varepsilon\Downarrow\mathscr{L}=\varepsilon$$

$$(m\cdot h)\Downarrow\mathscr{L}=m\Downarrow\mathscr{L}\cdot h\Downarrow\mathscr{L}$$

$$\emptyset\Downarrow\mathscr{L}=\emptyset$$

$$(H\cup\{s:h\})\Downarrow\mathscr{L}=\begin{cases}H\Downarrow\mathscr{L}\cup\{s:h\Downarrow\mathscr{L}\} & \textit{if } h\Downarrow\mathscr{L}\neq\varepsilon,\\ H\Downarrow\mathscr{L} & \textit{otherwise.}\end{cases}$$

Definition 3 ($\mathscr{L}$-Equality of Q-sets).
Two **Q***-sets H and K are $\mathscr{L}$-*equal*, written $H=_{\mathscr{L}}K$, if $H\Downarrow\mathscr{L}=K\Downarrow\mathscr{L}$.*

When reducing a configuration $(\nu\tilde{r})<P\,,\,H>$, we have to make sure that input prefixes in P "agree" with messages in H. This is assured by our type system given in Section 6.

A relation on processes is a *$\mathscr{L}$-bisimulation* if it preserves $\mathscr{L}$-equality of **Q**-sets at each step, starting from typable configurations:

Definition 4 ($\mathscr{L}$-Bisimulation on processes).
*A symmetric relation $\mathscr{R}\subseteq(\mathscr{P}r\times\mathscr{P}r)$ is a $\mathscr{L}$-*bisimulation* if $P_1\mathscr{R}P_2$ implies, for any pair of* **Q***-sets H_1 and H_2 such that $H_1=_{\mathscr{L}}H_2$ and $<P_1\,,\,H_1>$, $<P_2\,,\,H_2>$ are typable:*

If $<P_1\,,>H_1\longrightarrow(\nu\tilde{r})<P'_1\,,>H'_1$, then either $H'_1=_{\mathscr{L}}H_2$ and $P'_1\mathscr{R}P_2$, or there exist P'_2,H'_2 such that $<P_2\,,\,H_2>\longrightarrow^{*}(\nu\tilde{r})<P'_2\,,\,H'_2>$, where $H'_1=_{\mathscr{L}}H'_2$ and $P'_1\mathscr{R}P'_2$.

*Processes P_1,P_2 are $\mathscr{L}$-*bisimilar*, $P_1\simeq_{\mathscr{L}}P_2$, if $P_1\mathscr{R}P_2$ for some $\mathscr{L}$-bisimulation $\mathscr{R}$.*

Note that $\tilde{r}$ may be either empty or be a service name or a fresh session name s, and in the last case s cannot occur in P_2 and H_2 by Barendregt convention.

Intuitively, a transition that adds or removes a message with level in $\mathscr{L}$ must be simulated in one or more steps, producing the same effect on the **Q**-set, whereas a transition that only affects messages with level not in $\mathscr{L}$ may be simulated by inaction.

Definition 5 ($\mathscr{L}$-Security). *A program P is $\mathscr{L}$-*secure* if $P\simeq_{\mathscr{L}}P$.*

5 Examples of Information Flow Security in Sessions

In this section we illustrate the various kinds of flow that can occur in our calculus, through simple examples. Since we aim at justifying the introduction of security levels in the syntax (other than on values and participants), we shall initially omit levels in all other objects. In queues, we will use v^{ℓ} as a shorthand for $v^{\ell\downarrow\ell}$. For the sake of simplicity, we assume here just two security levels $\bot$ and $\top$ (also called low and high). In all examples, we suppose $H_1=\{s:(1,2,\mathsf{true}^{\top})\}$ and $H_2=\{s:(1,2,\mathsf{false}^{\top})\}$.

5.1. High input should not be followed by low actions. A simple example of insecure flow, which is not specific to our calculus but arises in all process calculi with values and a conditional construct, is the following (assuming session s has four participants):

$$\begin{array}{l} s[2]?(1,x^\top).\mathsf{if}\ x^\top\ \mathsf{then}\ s[2]!\langle 3,\mathsf{true}^\top\rangle.\mathbf{0}\ \mathsf{else}\ \mathbf{0} \\ \mid\ s[3]?(2,z^\top).s[3]!\langle 4,\mathsf{true}^\perp\rangle.\mathbf{0}\ \mid\ s[4]?(3,y^\perp).\mathbf{0} \end{array}$$

This process is insecure because, depending on the high value received for $x^\top$ on channel $s[2]$, that is, on whether the **Q**-set is H_1 or H_2, the low value $\mathsf{true}^\perp$ will be emitted or not on channel $s[3]$, leading to $H_1' = \{s : (3,4,\mathsf{true}^\perp)\} \neq_{\mathscr{L}} H_2' = \{s : \varepsilon\}$ if $\mathscr{L} = \{\perp\}$. This shows that a high input should not be followed by a low output. Note that the reverse is not true, since output is not blocking: if we swapped the polarities of input and output in the third participant (and adjusted them accordingly in the other participants), then the resulting process would be secure.

Let us point out that this process is not typable in a classical session type system, since the session types of the conditional branches are not the same. However, it would become typable if the second branch of the conditional were replaced by the deadlocked process $(\nu b)b[1](\beta_1).s[2]!\langle 3,\mathsf{true}^\top\rangle.\mathbf{0}$. The expert reader will notice that by adding to our type system the interaction typing of [2] (which enforces global progress) we would rule out also this second process. On the other hand, the interaction typing does not prevent deadlocks due to inverse session calls, as for instance:

$$\begin{array}{ll} \bar{b}[2] & \mid\ b[1](\beta_1).c[1](\gamma_1).s[2]!\langle 3,\mathsf{true}^\top\rangle.\mathbf{0} \\ \bar{c}[2] & \mid\ c[2](\gamma_2).b[2](\beta_2).\mathbf{0} \end{array}$$

Clearly, this deadlock could be used to implement the insecure flow in our example.

5.2. Need for levels on services. Consider the following process:

$$\begin{array}{l} s[2]?(1,x^\top).\mathsf{if}\ x^\top\ \mathsf{then}\ \bar{b}[2]\ \mathsf{else}\ \mathbf{0} \\ \mid\ b[1](\beta_1).\beta_1!\langle 2,\mathsf{true}^\perp\rangle.\mathbf{0}\ \mid\ b[2](\beta_2).\beta_2?(1,y^\perp).\mathbf{0} \end{array}$$

This process is insecure because, depending on the high value received for $x^\top$, it will initiate or not a session on service b, which performs a low value exchange. To rule out this kind of leak we annotate service names with security levels which are a lower bound for all the actions they execute. Then service b must be of level $\top$, since it appears in the branch of a $\top$-conditional, and hence it will not allow the output of the value $\mathsf{true}^\perp$.

5.3. Need for levels on selection and branching. Consider the following process:

$$\begin{array}{l} s[2]?(1,x^\top).\mathsf{if}\ x^\top\ \mathsf{then}\ s[2]\oplus\langle 3,\lambda\rangle.\mathbf{0}\ \mathsf{else}\ s[2]\oplus\langle 3,\lambda'\rangle.\mathbf{0} \\ \mid\ s[3]\&(2,\{\lambda : s[3]!\langle 4,\mathsf{true}^\perp\rangle.\mathbf{0}, \lambda' : s[3]!\langle 4,\mathsf{false}^\perp\rangle.\mathbf{0}\}) \\ \mid\ s[4]?(3,y^\perp).\mathbf{0} \end{array}$$

This process is insecure because a selection in one participant, which depends on a high value, causes the corresponding branching participant to emit two different low values. To prevent this kind of leak, the selection and branching operators will be annotated with a security level which is a lower bound for all actions executed in the branches.

5.4. Need for levels on delegated channels. Consider the following process:

$$\begin{array}{l} s[2]?(1,x^\top).\mathsf{if}\ x^\top\ \mathsf{then}\ s[2]!\langle\langle 3,s'[1]\rangle\rangle.s[2]!\langle\langle 4,s''[1]\rangle\rangle.\mathbf{0}\ \mathsf{else}\ s[2]!\langle\langle 3,s''[1]\rangle\rangle.s[2]!\langle\langle 4,s'[1]\rangle\rangle.\mathbf{0} \\ \mid\ s[3]?((2,\eta)).\eta!\langle 2,\mathsf{true}^\perp\rangle.\mathbf{0}\ \mid\ s[4]?((2,\eta')).\eta'!\langle 2,\mathsf{false}^\perp\rangle.\mathbf{0} \\ \mid\ s'[2]?(1,x^\perp).\mathbf{0}\ \mid\ s''[2]?(1,y^\perp).\mathbf{0} \end{array}$$

This process is insecure because, depending on the high value received for $x^\top$, the participants 3 and 4 of s will be delegated to participate in sessions s' and s'', or viceversa, feeding the queues of s' and s'' with different low values. This shows that delegation send and receive should also carry a level, which will be a lower bound for all actions executed in the receiving participant after the delegation.

5.5. Levels in queue messages. So far, we have identified which objects of the calculus need security levels, namely: values, service names, and the operators of selection, branching and delegation. We now discuss how levels are recorded into queue messages.

Values are recorded in the queues with both their level and their declassified level. The reason for recording also the declassified level is access control: the semantics does not allow a low input process to fetch a high value declassified to low. More formally, a value $v^{\top\downarrow\bot}$ in the queue can only be read by a process $s[\mathsf{q}]?^{\top}(\mathsf{p},x^{\bot}).P$. Concerning service names a^{ℓ}, the level ℓ guarantees that the session initiator and all the participants get started in a context of level $\ell' \leq \ell$ (see Example 5.2). Once the session is established, the name a^{ℓ} disappears and it is its global type (cf next section) that will ensure that all participants perform actions of levels greater than or equal to ℓ. As for the operators of branching/selection and delegation, they disappear after the reduction and their level is recorded respectively into labels and delegated channels within queue messages. This is essential since in this case the communication is asynchronous and occurs in two steps. Hence queue messages have the form $(\mathsf{p},\Pi,\vartheta)$, where ϑ is $v^{\ell\downarrow\ell'}$, λ^{ℓ} or $s[\mathsf{p}]^{\ell}$.

6 Type System

In this section we present our type system for secure sessions and state its properties. Just like process syntax, types will contain security levels.

Safety Global Types, Session Types, and Projections. A *safety global type* is a pair $\langle \mathsf{L}, G\rangle^{\ell}$, decorated with a security level ℓ, describing a service where:

- $\mathsf{L} : \{1,\ldots,n\} \to \mathscr{S}$ is a *safety mapping* from participants to security levels;
- G is a *global type*, describing the whole conversation scenario of an n-ary service;
- ℓ is the meet of all levels appearing in G, denoted by $\mathsf{M}(G)$.

The grammar of global types is:

$$\begin{array}{llll}
\text{Global} & G ::= \mathsf{p} \to \Pi : \langle U\rangle.G & \text{Exchange} & U ::= S^{\ell\downarrow\ell'} \mid T \mid \langle \mathsf{L}, G\rangle^{\ell} \\
& \quad\mid\ \mathsf{p} \to \Pi : \{\lambda_i : G_i\}^{\ell}_{i\in I} & \text{Sorts} & S ::= \mathsf{bool} \mid \ldots \\
& \quad\mid\ \mathsf{p} \Rsh \delta.G & & \\
& \quad\mid\ \mu\mathbf{t}.G \mid \mathbf{t} \mid \mathsf{end} & &
\end{array}$$

The type $\mathsf{p} \to \Pi : \langle U\rangle.G$ says that participant p multicasts a message of type U to all participants in Π and then the interactions described in G take place. *Exchange types U* may be *sort* types $S^{\ell\downarrow\ell'}$ for values (base types decorated with a declassification $\ell\downarrow\ell'$), *session types T* for channels (defined below), or safety global types for services. If $U = T$, then Π is a singleton $\{\mathsf{q}\}$. We use S^{ℓ} as short for $S^{\ell\downarrow\ell}$, called a *trivial declassification*. Type $\mathsf{p} \to \Pi : \{\lambda_i : G_i\}^{\ell}_{i\in I}$, where $\ell = \bigsqcap_{i\in I}\mathsf{M}(G_i)$, says that participant p multicasts one of the labels λ_i to the participants in Π. If λ_j is sent, interactions described in G_j take place. Type $\mathsf{p} \Rsh \delta.G$ says that the role of p is delegated to another participant; this construct does not appear in the original global types of [11]. It is needed here to "mark" the delegated part of the type, which is discharged when calculating its join (see below).

Type $\mu\mathbf{t}.G$ is a recursive type, where the type variable $\mathbf{t}$ is guarded in the standard way. In the grammar of exchange types, we suppose that G does not contain free type variables. Type end represents the termination of a session. While global types represent the whole session protocol, *session types* correspond to the communication actions, representing each participant's contribution to the session.
As for $\mathsf{M}(G)$, we denote by $\mathsf{M}(T)$ the meet of all security levels appearing in T.

$$\begin{array}{lllrlr}
\text{Session} & T ::= & !\langle \Pi, S^{\ell\downarrow\ell'}\rangle;T & \textit{send} & \mid\ ?(\mathsf{p}, S^{\ell\downarrow\ell'});T & \textit{receive} \\
 & \mid & !^{\ell}\langle \mathsf{q}, T\rangle;T' & \textit{delsend} & \mid\ ?^{\ell}(\mathsf{p}, T);T' & \textit{delreceive} \\
 & \mid & \oplus^{\ell}\langle \Pi, \{\lambda_i : T_i\}_{i\in I}\rangle & \textit{selection} & \mid\ \&^{\ell}(\mathsf{p}, \{\lambda_i : T_i\}_{i\in I}) & \textit{branching} \\
 & \mid & \mu\mathbf{t}.T & \textit{recursive} & \mid\ \mathbf{t} & \textit{variable} \\
 & \mid & \Rsh\delta;T & \textit{delegation} & \mid\ \mathsf{end} & \textit{end}
\end{array}$$

The *send* type $!\langle \Pi, S^{\ell\downarrow\ell'}\rangle;T$ expresses the sending to all participants in Π of a value of type S, of level ℓ declassified to ℓ', followed by the communications described in T. The *delsend* type $!^{\ell}\langle \mathsf{q}, T\rangle;T'$, where $\ell = \mathsf{M}(T)$, says that a channel of type T is sent to participant q, and then the protocol specified by T' takes place. The *selection* type $\oplus^{\ell}\langle \Pi, \{\lambda_i : T_i\}_{i\in I}\rangle$, where $\ell = \bigsqcap_{i\in I} \mathsf{M}(T_i)$, represents the transmission to all participants in Π of a label λ_j in $\{\lambda_i \mid i \in I\}$, followed by the communications described in T_j. The *delegation* type $\Rsh\delta;T$, says that the communications described in T will be delegated to another agent. The *receive*, *delreceive* and *branching* types are dual to the *send*, *delsend*, and *selection* ones. The type system will assure that $\ell' \leq \mathsf{M}(T)$ in type $?(\mathsf{p}, S^{\ell\downarrow\ell'});T$, that $\ell \leq \mathsf{M}(T')$ in type $?^{\ell}(\mathsf{p}, T);T'$ and that $\ell = \bigsqcap_{i\in I} \mathsf{M}(T_i)$ in type $\&^{\ell}(\mathsf{p}, \{\lambda_i : T_i\}_{i\in I})$. In all cases, the need for the security level ℓ is motivated by one of the examples in Section 5.

The relation between global types and session types is formalised by the notion of projection [11]. The *projection of G onto* q, denoted $(G \upharpoonright \mathsf{q})$, gives participant q's view of the protocol described by G. For example the projection of $G = \mathsf{p} \to \mathsf{p}' : \langle T\rangle.G'$ on q is the following, assuming $\ell = \mathsf{M}(T)$:

$$(\mathsf{p} \to \mathsf{p}' : \langle T\rangle.G') \upharpoonright \mathsf{q} = \begin{cases} !^{\ell}\langle \mathsf{p}', T\rangle; (G' \upharpoonright \mathsf{q}) & \text{if } \mathsf{q} = \mathsf{p}, \\ ?^{\ell}(\mathsf{p}, T); (G' \upharpoonright \mathsf{q}) & \text{if } \mathsf{q} = \mathsf{p}', \\ G' \upharpoonright \mathsf{q} & \text{otherwise} \end{cases}$$

Well-formedness of safety global types. To formulate the well-formedness condition for safety global types, we define the join $\mathsf{J}(T)$ of a session type T. Intuitively, while $\mathsf{M}(T)$ is needed for secure information flow, $\mathsf{J}(T)$ will be used for access control. Recall from Section 2 our access control policy, requiring that participants in a session only read data of level less than or equal to their own level. This motivates our (slightly non standard) definition of join: in short, $\mathsf{J}(T)$ is the join of all the security levels decorating the input constructs in T (*receive, delreceive, branching*). Moreover, unlike $\mathsf{M}(T)$, $\mathsf{J}(T)$ forgets the delegated part of T.

This leads to the following condition of well-formedness for safety global types, where $dom(\mathsf{L})$ denotes the domain of L:

A safety global type $\langle \mathsf{L}, G\rangle^{\ell}$ is well formed if for all $\mathsf{p} \in dom(\mathsf{L})$*:* $\mathsf{L}(\mathsf{p}) \geq \mathsf{J}(G \upharpoonright \mathsf{p})$.

Henceforth we shall only consider well-formed safety global types.

Typing expressions. The typing judgments for expressions are of the form:

$$\Gamma \vdash e : S^{\ell}$$

where Γ is the *standard environment* which maps variables to sort types with trivial declassification, services to safety global types, and process variables to pairs of sort types with trivial declassification and session types. Formally, we define:

$$\Gamma ::= \emptyset \mid \Gamma, x^{\ell} : S^{\ell'} \mid \Gamma, a^{\ell} : \langle \mathsf{L}, G \rangle^{\ell'} \mid \Gamma, X : S^{\ell}\ T$$

assuming that we can write $\Gamma, x^{\ell} : S^{\ell'}$ (respectively $\Gamma, a^{\ell} : \langle \mathsf{L}, G \rangle^{\ell'}$ and $\Gamma, X : S^{\ell}\ T$) only if x^{ℓ} (respectively a^{ℓ} and X) does not belong to the domain of Γ. An environment Γ is well formed if $x^{\ell} : S^{\ell'} \in \Gamma$ implies $\ell' = \ell$ and $a^{\ell} : \langle \mathsf{L}, G \rangle^{\ell'} \in \Gamma$ implies that $\ell' = \ell$ and G is well formed. Hence, if Γ is well formed, $a^{\ell} : \langle \mathsf{L}, G^{\ell} \rangle \in \Gamma$ implies $\ell = \mathsf{M}(G)$. In the following we will only consider well-formed environments.

We type values by decorating their type with their security level, and names according to Γ:

$$\Gamma \vdash \ \mathsf{true}^{\ell}, \mathsf{false}^{\ell} : \mathsf{bool}^{\ell} \qquad \Gamma, u : S^{\ell} \vdash u : S^{\ell} \qquad \lfloor \textsc{Name} \rfloor$$

We type expressions by decorating their type with the join of the security levels of the variables and values they are built from.

Typing processes. The typing judgments for processes are of the form:

$$\Gamma \vdash_{\ell} P \triangleright \Delta$$

where Δ is the *process environment* which associates session types with channels:

$$\Delta ::= \emptyset \mid \Delta, c : T$$

We decorate the derivation symbol $\vdash$ with the security level ℓ inferred for the process: this level is a lower bound for the actions and communications performed in the process.

Let us now present some selected typing rules for processes.

– Rule $\lfloor \textsc{SubS} \rfloor$ allows the security level inferred for a process to be decreased.

$$\frac{\Gamma \vdash_{\ell} P \triangleright \Delta \quad \ell' \leq \ell}{\Gamma \vdash_{\ell'} P \triangleright \Delta} \lfloor \textsc{SubS} \rfloor$$

– In rule $\lfloor \textsc{MInit} \rfloor$, the standard environment must associate with the identifier u a safety global type. The premise matches the number of participants in the domain of L with the number declared by the initiator. The emptiness of the process environment in the conclusion specifies that there is no further communication behaviour after the initiator.

$$\frac{dom(\mathsf{L}) = \{1, \ldots, n\}}{\Gamma, u : \langle \mathsf{L}, G \rangle^{\ell} \vdash_{\ell} \bar{u}[n] \triangleright \emptyset} \lfloor \textsc{MInit} \rfloor$$

– In rule $\lfloor \textsc{MAcc} \rfloor$, the standard environment must also associate with u a safety global type. The premise guarantees that the type of the continuation P in the p-th participant is the p-th projection of the global type G of u.

$$\frac{\Gamma, u : \langle \mathsf{L}, G \rangle^{\ell} \vdash_{\ell} P \triangleright \Delta, \alpha : G \restriction \mathtt{p}}{\Gamma, u : \langle \mathsf{L}, G \rangle^{\ell} \vdash_{\ell} u[\mathtt{p}](\alpha).P \triangleright \Delta} \lfloor \textsc{MAcc} \rfloor$$

Concerning security levels, in rule $\lfloor\textsc{MAcc}\rfloor$ we check that the continuation process P conforms to the security level ℓ associated with the service name u. Note that this condition does not follow from well-formedness of environments, since the process P may participate in other sessions, but it is necessary to avoid information leaks. For example, without this condition we could type

$$\begin{array}{lll} \bar{a}^{\top}[2] & \mid & a^{\top}[1](\alpha_1).\alpha_1!\langle 2,\mathsf{true}^{\top}\rangle.\mathbf{0} \mid a^{\top}[2](\alpha_2).\alpha_2?(1,x^{\top}).\mathsf{if}\ x^{\top}\ \mathsf{then}\ \bar{b}^{\top}[2]\ \mathsf{else}\ \mathbf{0} \\ & \mid & b^{\top}[1](\beta_1).c^{\bot}[1](\gamma_1).\gamma_1!\langle 2,\mathsf{true}^{\bot}\rangle.\mathbf{0} \mid b^{\top}[2](\beta_2).\mathbf{0} \\ \bar{c}^{\bot}[2] & \mid & c[2](\gamma_2).\gamma_2?(1,y^{\bot}).\mathbf{0} \end{array}$$

– In rule $\lfloor\textsc{Send}\rfloor$, the first hypothesis binds expression e with type S^{ℓ}, where ℓ is the join of all variables and values in e. The second hypothesis imposes typability of the continuation of the output with security level ℓ''. The third hypothesis relates levels ℓ, ℓ'' and ℓ' (the level to which e will be declassified), preserving the invariant that ℓ'' is a lower bound for all security levels of the actions in the process.

$$\frac{\Gamma \vdash e : S^{\ell} \qquad \Gamma \vdash_{\ell''} P \triangleright \Delta, c:T \qquad \ell'' \le \ell' \le \ell}{\Gamma \vdash_{\ell''} c!^{\ell'}\langle \Pi, e\rangle.P \triangleright \Delta, c: !\langle \Pi, S^{\ell\downarrow\ell'}\rangle;T}\ \lfloor\textsc{Send}\rfloor$$

Note that the hypothesis $\ell'' \le \ell' \le \ell$ is not really constraining, since P can always be downgraded to ℓ'' using rule $\lfloor\textsc{SubS}\rfloor$ and $\ell' \le \ell$ follows from well-formedness of $S^{\ell\downarrow\ell'}$.

– Rule $\lfloor\textsc{Rcv}\rfloor$ is the dual of rule $\lfloor\textsc{Send}\rfloor$, but it is more restrictive in that it requires the continuation P to be typable with *exactly* the level ℓ':

$$\frac{\Gamma, x^{\ell'} : S^{\ell'} \vdash_{\ell'} P \triangleright \Delta, c:T \qquad \ell' \le \ell}{\Gamma \vdash_{\ell'} c?^{\ell}(\mathsf{p}, x^{\ell'}).P \triangleright \Delta, c:?(\mathsf{p}, S^{\ell\downarrow\ell'});T}\ \lfloor\textsc{Rcv}\rfloor$$

Notice for instance that we cannot type the reception of a $\top$ value followed by a $\bot$ action. On the other hand we can type the reception of a $\top \downarrow \bot$ value followed by a $\bot$ action. For instance, in our introductory example of Section 2, rule [Rcv] allows the delegation send in process B to be decorated by $\bot$: this is essential for the typability of both the process B and the session b between S and B.

– Rule $\lfloor\textsc{If}\rfloor$ requires that the two branches of a conditional be typed with the same process environment, and with the same security level as the tested expression.

$$\frac{\Gamma \vdash e : \mathsf{bool}^{\ell} \qquad \Gamma \vdash_{\ell} P \triangleright \Delta \qquad \Gamma \vdash_{\ell} Q \triangleright \Delta}{\Gamma \vdash_{\ell} \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta}\ \lfloor\textsc{If}\rfloor$$

We say that a process P is typable in Γ if $\Gamma \vdash_{\ell} P \triangleright \Delta$ holds for some ℓ, Δ.

Typing queues and Q-sets. *Message types* represent the messages contained in queues.

$$\begin{array}{rcll} \text{Message } \mathbf{T} & ::= & !\langle \Pi, S^{\ell\downarrow\ell'}\rangle & \textit{message value send} \\ & \mid & !^{\ell}\langle \mathsf{q}, T\rangle & \textit{message delegation} \\ & \mid & \oplus^{\ell}\langle \Pi, \lambda\rangle & \textit{message selection} \\ & \mid & \mathbf{T};\mathbf{T}' & \textit{message sequence} \end{array}$$

Message types are very close to the *send, delsend, selection* session types, hence we shall not dwell on them. Let us just mention the associativity of the construct $\mathbf{T};\mathbf{T}'$.

Typing judgments for queues have the shape

$$\Gamma \vdash s : h \triangleright \Theta$$

where Θ is a *queue environment* associating message types with channels.

Example: we can derive $\vdash s : (2, \{1,3\}, \mathsf{ok}^{\top}) \triangleright \{s[2] : \oplus^{\top}\langle \{1,3\}, \mathsf{ok}\rangle\}$.

Typing judgments for **Q**-sets have the shape:

$$\Gamma \vdash_{\Sigma} H \triangleright \Theta$$

where Σ is the set of session names which occur free in H.

Typing configurations. Typing judgments for runtime configurations C have the form:

$$\Gamma \vdash_{\Sigma} C \triangleright < \Delta \diamond \Theta >$$

They associate with a configuration the environments Δ and Θ mapping channels to session and message types respectively. We call $< \Delta \diamond \Theta >$ a *configuration environment*.

A *configuration type* is a session type, or a message type, or a message type followed by a session type:

$$\begin{array}{rlr} \text{Configuration } \mathscr{T} ::= & T & \textit{session} \\ | & \mathbf{T} & \textit{message} \\ | & \mathbf{T}; T & \textit{continuation} \end{array}$$

An example of configuration type is:

$$\oplus^{\perp}\langle \{1,3\}, \mathsf{ok}\rangle; !\langle \{3\}, \mathsf{String}^{\top}\rangle; ?(3, \mathsf{Number}^{\perp}); \mathsf{end}$$

A configuration is *initial* if the process is closed, it does not use runtime syntax and the **Q**-set is empty. It is easy to check that for typable initial configurations the set of session names and the process and queue environments are all empty.

Since channels with roles occur both in processes and in queues, a configuration environment associates configuration types with channels, in notation $< \Delta \diamond \Theta >(c)$. Configuration types can be projected on a participant p. We write $\mathscr{T} \restriction \mathsf{p}$ to denote the projection of the type $\mathscr{T}$ on the participant p. We also define a duality relation $\bowtie$ between projections of configuration types, which holds when opposite communications are offered (input/output, selection/branching). The above definitions are needed to state coherence of configuration environments. Informally, this holds when the inputs and the branchings offered by the process agree both with the outputs and the selections offered by the process and with the messages in the queues. More formally:

Definition 6. *A configuration environment $< \Delta \diamond \Theta >$ is coherent if $s[\mathsf{p}] \in dom(\Delta) \cup dom(\Theta)$ and $s[\mathsf{q}] \in dom(\Delta) \cup dom(\Theta)$ imply*

$$< \Delta \diamond \Theta > (s[\mathsf{p}]) \restriction \mathsf{q} \bowtie < \Delta \diamond \Theta > (s[\mathsf{q}]) \restriction \mathsf{p}.$$

Typing rules assure that configurations are always typed with coherent environments.

Since process and queue environments represent future communications, by reducing processes we get different configuration environments. This is formalised by the notion of reduction of configuration environments, denoted by $< \Delta \diamond \Theta > \Rightarrow < \Delta' \diamond \Theta' >$.

We say that a queue is *generated by service a*, or *a-generated*, if it is created by applying rule [Link] to the parallel composition of a's participants and initiator.

We are now able to state our main results, namely type preservation under reduction and the soundness of our type system for both access control and noninterference. In Theorem 2, we will use the function $lev_{\uparrow}(\vartheta)$ defined in Section 4 (Definition 1).

Theorem 1 (Subject Reduction). *Suppose* $\Gamma \vdash_{\Sigma} C \triangleright < \Delta \diamond \Theta >$ *and* $C \longrightarrow^{*} C'$. *Then* $\Gamma \vdash_{\Sigma} C' \triangleright < \Delta' \diamond \Theta' >$ *with* $< \Delta \diamond \Theta > \;\Rightarrow\; < \Delta' \diamond \Theta' >$.

Theorem 2 (Access Control)
Let C *be an initial configuration, and suppose* $\Gamma \vdash_{\emptyset} C \triangleright < \emptyset \diamond \emptyset >$ *for some standard environment* Γ *such that* $a^{\ell} : \langle \mathsf{L}, G \rangle^{\ell} \in \Gamma$. *If* $C \longrightarrow^{*} (\nu s)C'$, *where the queue of name* s *in* C' *is a-generated and contains the message* $(\mathtt{p}, \mathtt{q}, \vartheta)$, *then* $lev_{\uparrow}(\vartheta) \leq \mathsf{L}(\mathtt{q})$.

Theorem 3 (Noninterference). *If* P *is typable, then* $P \simeq_{\mathscr{L}} P$ *for all down-closed* $\mathscr{L}$.

7 Conclusion and Future Work

In this work, we have investigated the integration of security requirements into session types. Interestingly, there appears to be an influence of session types on security.

For instance, it is well known that one of the causes of insecure information flow in a concurrent scenario is the possibility of different termination behaviours in the branches of a high conditional. In our calculus, we may distinguish three termination behaviours: (proper) termination, deadlock and divergence. Now, the classical session types of [20] already exclude some combinations of these behaviours in conditional branches. For instance, a non-trivial divergence (whose body contains some communication actions) in one branch cannot coexist with a non-trivial termination in the other branch. Moreover, session types prevent *local deadlocks* due to a bad matching of the communication behaviours of participants in the same session. By adding to classical session types the interaction typing of [2], we would also exclude most of the *global deadlocks* due to a bad matching of the protocols of two interleaved sessions. However, this typing does not prevent deadlocks due to inverse session calls. We plan to study a strengthening of interaction typing that would rule out also this kind of deadlock. This would allow us to simplify our type system by removing our constraint in the typing rule for input.

The form of declassification considered in this work is admittedly quite simple. However, it already illustrates the connection between declassification and access control, since a declassified value may only be received by a participant whose level is greater than or equal to the original level of the value. This means that declassification is constrained by the access control policy, as in [5]. We plan to extend declassification also to data which are not received from another participant, by allowing declassification of a tested expression, as in this variant of the B process of our example in Section 2:

$$\mathsf{B}' \;=\; \ldots\ \mathsf{if}\ \{cc^{\perp} = secret^{\top}\}^{\top\downarrow\perp}\ \mathsf{then}\ \beta_1 \oplus^{\perp} \langle 2, \mathsf{ok}\rangle.\mathbf{0}\ \mathsf{else}\ \beta_1 \oplus^{\perp} \langle 2, \mathsf{ko}\rangle.\mathbf{0}$$

Again, this declassification would be controlled by requiring B' to have level $\top$.

Acknowledgments. We would like to thank Nobuko Yoshida for her encouragement to engage in this work, and the anonymous referees for helpful feedback.

References

1. Almeida Matos, A., Boudol, G.: On Declassification and the Non-Disclosure Policy. Journal of Computer Security 17, 549–597 (2009)
2. Bettini, L., Coppo, M., D'Antoni, L., De Luca, M., Dezani-Ciancaglini, M., Yoshida, N.: Global Progress in Dynamically Interleaved Multiparty Sessions. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 418–433. Springer, Heidelberg (2008)
3. Bhargavan, K., Corin, R., Deniélou, P.-M., Fournet, C., Leifer, J.J.: Cryptographic Protocol Synthesis and Verification for Multiparty Sessions. In: Proc. CSF 2009, pp. 124–140. IEEE Computer Society, Los Alamitos (2009)
4. Boreale, M., Bruni, R., Nicola, R., Loreti, M.: Sessions and Pipelines for Structured Service Programming. In: Barthe, G., de Boer, F.S. (eds.) FMOODS 2008. LNCS, vol. 5051, pp. 19–38. Springer, Heidelberg (2008)
5. Boudol, G., Kolundzija, M.: Access Control and Declassification. In: Proc. Computer Network Security. CCIS, vol. 1, pp. 85–98. Springer, Heidelberg (2007)
6. Denning, D.E.R.: Cryptography and Data Security. Addison-Wesley, Reading (1982)
7. Dezani-Ciancaglini, M., de'Liguoro, U.: Sessions and Session Types: An Overview. In: Laneve, C. (ed.) WSFM 2010. LNCS, vol. 6194, pp. 1–28. Springer, Heidelberg (2010)
8. Focardi, R., Gorrieri, R.: Classification of Security Properties (Part I: Information Flow). In: Focardi, R., Gorrieri, R. (eds.) FOSAD 2000. LNCS, vol. 2171, pp. 331–396. Springer, Heidelberg (2001)
9. Goguen, J.A., Meseguer, J.: Security Policies and Security Models. In: Proc. IEEE Symposium on Security and Privacy, pp. 11–20. IEEE Computer Society, Los Alamitos (1982)
10. Honda, K., Yoshida, N.: A Uniform Type Structure for Secure Information Flow. In: Proc. POPL 2002, pp. 81–92. ACM Press, New York (2002)
11. Honda, K., Yoshida, N., Carbone, M.: Multiparty Asynchronous Session Types. In: Proc. POPL 2008, pp. 273–284. ACM Press, New York (2008)
12. Kobayashi, N.: Type-Based Information Flow Analysis for the Pi-Calculus. Acta Informatica 42(4-5), 291–347 (2005)
13. Kolundžija, M.: Security Types for Sessions and Pipelines. In: Bruni, R., Wolf, K. (eds.) WSFM 2008. LNCS, vol. 5387, pp. 175–190. Springer, Heidelberg (2009)
14. Milner, R.: Communicating and Mobile Systems: the Pi-Calculus. CUP, Cambridge (1999)
15. Planul, J., Corin, R., Fournet, C.: Secure Enforcement for Global Process Specifications. In: Proc. CONCUR 2009. LNCS, vol. 5710, pp. 511–526. Springer, Heidelberg (2009)
16. Sabelfeld, A., Myers, A.C.: Language-Based Information-Flow Security. IEEE Journal on Selected Areas in Communications 21(1), 5–19 (2003)
17. Sabelfeld, A., Sands, D.: Probabilistic Noninterference for Multi-threaded Programs. In: Proc. CSFW 2000, pp. 200–214. IEEE Computer Society, Los Alamitos (2000)
18. Sabelfeld, A., Sands, D.: Dimensions and Principles of Declassification. In: Proc. CSFW 2005. IEEE Computer Society, Los Alamitos (2005)
19. Smith, G., Volpano, D.: Secure Information Flow in a Multi-threaded Imperative Language. In: Proc. POPL 1998, pp. 355–364. ACM Press, New York (1998)
20. Takeuchi, K., Honda, K., Kubo, M.: An Interaction-based Language and its Typing System. In: Halatsis, C., Philokyprou, G., Maritsas, D., Theodoridis, S. (eds.) PARLE 1994. LNCS, vol. 817, pp. 398–413. Springer, Heidelberg (1994)
21. Volpano, D., Irvine, C., Smith, G.: A Sound Type System for Secure Flow Analysis. Journal of Computer Security 4(2,3), 167–187 (1996)

Simulation Distances[⋆]

Pavol Černý, Thomas A. Henzinger, and Arjun Radhakrishna

IST Austria

Abstract. Boolean notions of correctness are formalized by preorders on systems. Quantitative measures of correctness can be formalized by real-valued distance functions between systems, where the distance between implementation and specification provides a measure of "fit" or "desirability." We extend the simulation preorder to the quantitative setting, by making each player of a simulation game pay a certain price for her choices. We use the resulting games with quantitative objectives to define three different simulation distances. The *correctness distance* measures how much the specification must be changed in order to be satisfied by the implementation. The *coverage distance* measures how much the implementation restricts the degrees of freedom offered by the specification. The *robustness distance* measures how much a system can deviate from the implementation description without violating the specification. We consider these distances for safety as well as liveness specifications. The distances can be computed in polynomial time for safety specifications, and for liveness specifications given by weak fairness constraints. We show that the distance functions satisfy the triangle inequality, that the distance between two systems does not increase under parallel composition with a third system, and that the distance between two systems can be bounded from above and below by distances between abstractions of the two systems. These properties suggest that our simulation distances provide an appropriate basis for a quantitative theory of discrete systems. We also demonstrate how the robustness distance can be used to measure how many transmission errors are tolerated by error correcting codes.

1 Introduction

Standard verification systems return a boolean answer that indicates whether a system satisfies its specification. However, not all correct implementations are equally good, and not all incorrect implementations are equally bad. There is thus a natural question whether it is possible to extend the standard specification frameworks and verification algorithms to capture a finer and more quantitative view of the relationship between specifications and systems.

We focus on extending the notion of simulation to the quantitative setting. For reactive systems, the standard correctness requirement is that all executions

[⋆] This work was partially supported by the European Union project COMBEST and the European Network of Excellence ArtistDesign.

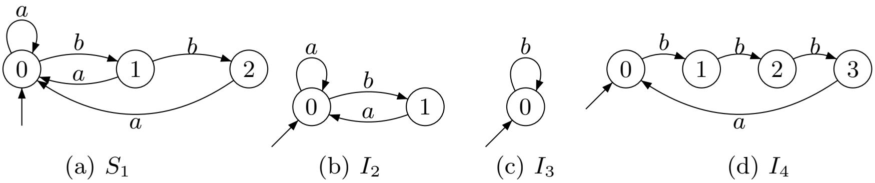
P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 253–268, 2010.

of an implementation have to be allowed by the specification. Requiring that the specification simulates the implementation is a stricter condition, but it is computationally less expensive to check. The simulation relation defines a preorder on systems. We extend the simulation preorder to a distance function that given two systems, returns a real-valued distance between them.

Let us consider the definition of simulation of an implementation I by a specification S as a two-player game, where Player 1 (the implementation) chooses moves (transitions) and Player 2 (the specification) tries to match each move. The goal of Player 1 is to prove that simulation does not hold, by driving the game into a state from which Player 2 cannot match the chosen move; the goal of Player 2 is to prove that there exists a simulation relation, by playing the game forever. In order to extend this definition to capture how "good" (or how "bad") the simulation is, we make the players pay a certain price for their choices. The goal of Player 1 is then to maximize the cost of the game, and the goal of Player 2 is to minimize it. The cost is given by an objective function, such as the limit average of transition prizes. For example, for incorrect implementations, i.e., those for which the specification S does not simulate the implementation I, we might be interested in how often the specification (Player 2) cannot match an implementation move. We formalize this using a game with a limit-average objective between modified systems. The specification is allowed to "cheat," by following a non-existing transition, while the implementation is left unmodified. More precisely, the specification is modified by giving the transitions from the original system a weight of 0, and adding new "cheating" transitions with a non-zero positive weight. As Player 2 is trying to minimize the value of the game, she is motivated not to cheat. The value of the game measures how often the specification can be forced to cheat by the implementation, that is, how often the implementation violates the specification (i.e., commits an error) in the worst case. We call this distance function *correctness*.

Let us consider the examples in Figure 1. We take the system S_1 as the specification. The specification allows at most two symbols b to be output in the row. Now let us consider the two incorrect implementations I_3 and I_4. The implementation I_3 outputs an unbounded number of b's in a row, while the implementation I_4 can output three b's in a row. The specification S_1 will thus not be able to simulate either I_3 or I_4, but I_4 is a "better" implementation in the sense that it violates the requirement to a smaller degree. We capture this by allowing S_1 to cheat in the simulation game by taking an existing edge while outputting a different symbol. When simulating the system I_3, the specification S_1 will have to output a b when taking the edge from state 2 to state 0. This cheating transition will be taken every third move while simulating I_3. The correctness distance from S_1 to I_3 will therefore be 1/3. When simulating I_4, the specification S_1 needs to cheat only one in four times—this is when I_4 takes a transition from its state 2 to state 3. The distance from S_1 to I_4 will be 1/4.

Considering the implementation I_2 from Figure 1, it is easy to see that it is correct with respect to the specification S_1. The correctness distance would thus be 0. However, it is also easy to see that I_2 does not include all behaviors allowed



(a) S_1 (b) I_2 (c) I_3 (d) I_4

Fig. 1. Example Systems

by S_1. Our second distance function, *coverage*, is the dual of the correctness distance. It measures how many of the behaviors allowed by the specification are actually implemented by the implementation. This distance is obtained as the value for the implementation in a game in which I is required to simulate S, with the implementation being allowed to cheat. Our third distance function is called *robustness*. It measures how robust the implementation I is with respect to the specification S in the following sense: we measure how often the implementation can make an unexpected error (i.e., it performs a transition not present in its transition relation), with the resulting behavior still being accepted by the specification. Unexpected errors could be caused, for example, by a hardware problem, by a wrong environment assumption, or by a malicious attack. Robustness measures how many such unexpected errors are tolerated.

In addition to safety specifications, we consider liveness specifications given by weak (Büchi) fairness constraints or strong (Streett) fairness constrains. In order to define distances to liveness specifications, the notion of quantitative simulation is extended to *fair* quantitative simulation. We study variations of the correctness, coverage, and robustness distances using limit-average and discounted objective functions. Limit-average objectives measure the long-run frequency of errors, whereas discounted objectives count the number of errors and give more weight to earlier errors than later ones.

The correctness, coverage, and robustness distances can be calculated by solving the value problem in the corresponding games. Without fairness requirements, we obtain limit-average games or discounted games with constant weights. The values of such games can be computed in polynomial time [20]. We obtain polynomial complexity also for distances between systems with weak-fairness constraints, whereas for strong-fairness constrains, the best known algorithms require exponential time.

We present composition and abstraction techniques that are useful for computing and approximating simulation distances between large systems. Finally, we present an application of the robustness distance. We consider error correction systems for transmitting data over noisy channels. Three implementations based on the Hamming code, triple modular redundancy, and no error correction with different robustness properties are analyzed.

Related work. Weighted automata [4,10] provide a way to assign values to words, and to languages defined by finite-state systems. Distances between systems can be defined using weighted automata, analogically to boolean language inclusion. However, the complexity of computation of such distance is not known [4]. Our solution of using a quantitative version of simulation games corresponds in the boolean case to the choice of using simulation instead of language inclusion. There have been several attempts to give a mathematical semantics to reactive processes which is based on quantitative metrics rather than boolean preorders [18,6]. In particular for probabilistic processes, it is natural to generalize bisimulation relations to bisimulation metrics [9,19], and similar generalizations can be pursued if quantities enter not through probabilities but through discounting [7] or continuous variables [2] (this work uses the Skorohod metric on continuous behaviors to measure the distance between hybrid systems). We consider distances between purely discrete (nonprobabilistic, untimed) systems, and our distances are directed rather than symmetric (based on simulation rather than bisimulation). Software metrics measure properties such as lines of code, depth of inheritance (in an object-oriented language), number of bugs in a module or the time it took to discover the bugs (see for example [12,16]). These functions measure syntactic properties of the source code, and are fundamentally different from our distance functions that capture the difference in the behavior (semantics) of programs.

2 Quantitative Simulation Games

Transition Systems. A *transition system* is a tuple $\langle S, \Sigma, E, s_0 \rangle$ where S is a finite set of states, Σ is a finite alphabet, $E \subseteq S \times \Sigma \times S$ is a set of labeled transitions, and s_0 is the initial state. We require that for every $s \in S$, there exists a transition from s. The set of all transition systems is denoted by $\mathcal{S}$. A *weighted transition system* is a transition system along with a weight function v from E to $\mathbb{Q}$. A run in a transition system T is an infinite path $\rho = \rho_0 \sigma_0 \rho_1 \sigma_1 \rho_2 \sigma_2 \ldots \in (S \cdot \Sigma)^\omega$ where $\rho_0 = s_0$ and for all i, $(\rho_i, \sigma_i, \rho_{i+1}) \in E$.

Fairness Conditions. A *Büchi (weak fairness) condition* for a (weighted) transition system is set of states $F \subseteq S$. Given a Büchi condition F and a run $\rho = \rho_0 \sigma_0 \rho_1 \sigma_1 \ldots$ of a transition system, the run ρ is *fair* iff $\forall n \geq 0 : (\exists i > n : \rho_i \in F)$. A *Streett (strong fairness) condition* for a (weighted) transition system is a set of request-response pairs $F = \{\langle E_1, F_1 \rangle, \langle E_2, F_2 \rangle, \ldots, \langle E_d, F_d \rangle\}$ where each $E_i, F_i \in 2^S$. Given a Streett condition, a run $\rho = \rho_0 \sigma_0 \rho_1 \sigma_1 \ldots$ is *fair* iff $\forall k \leq d : \big((|\{i \mid \rho_i \in E_k\}| = \infty) \Rightarrow (|\{i \mid \rho_i \in F_k\}| = \infty) \big)$. We denote a transition system A with a fairness condition F as A^F.

Game Graphs. A *game graph* G is a tuple $\langle S, S_1, S_2, \Sigma, E, s_0 \rangle$ where S, Σ, E and s_0 are as in transition systems and (S_1, S_2) is a partition of S. The choice of the next state is made by Player 1 (Player 2) when the current state is in S_1 (respectively, S_2). A *weighted game graph* is a game graph along with a weight function v from E to $\mathbb{Q}$. A run in the game graph G is called a *play*. The set of all plays is denoted by Ω.

When the two players represent the choices internal to a system, we call the game graph an *alternating transition system*. We only consider alternating transition systems where the transitions from Player 1 states go only to Player 2 states and vice-versa. We use A^F to denote an alternating transition system A with fairness condition F.

Strategies. Given a game graph G, a *strategy* for Player 1 is a function $\pi : (S \cdot \Sigma)^* S_1 \rightarrow S \times \Sigma$ such that $\forall s_0 \sigma_0 s_1 \sigma_1 \ldots s_i \in (S \cdot \Sigma)^* S_1$, we have that if $\pi(s_0 \sigma_0 s_1 \sigma_1 \ldots s_i) = (s, \sigma)$, then $(s_i, \sigma, s) \in E$. A strategy for Player 2 is defined similarly. The set of all strategies for Player p is denoted by Π_p. A play $\rho = \rho_0 \sigma_0 \rho_1 \sigma_1 \rho_2 \sigma_2 \ldots$ *conforms* to a player p strategy π if $\forall i \geq 0 : (\rho_i \in S_p \implies : (\rho_{i+1}, \sigma_{i+1}) = \pi(\rho_0 \sigma_0 \rho_1 \sigma_1 \ldots \rho_i))$. The *outcome* of strategies π_1 and π_2 is the unique play $out(\pi_1, \pi_2)$ that conforms to both π_1 and π_2.

Two restricted notions of a strategy are sufficient for many classes of games. A *memoryless strategy* is one where the value of the strategy function depends solely on the last state in the history, whereas a *finite-memory strategy* is one where the necessary information about the history can be summarized by a finite amount of information.

Games and Objectives. A *game* is a game graph and a boolean or quantitative objective. A *boolean objective* is a function $\Phi : \Omega \rightarrow \{0, 1\}$ and the goal of Player 1 in a game with objective Φ is to choose a strategy so that, no matter what Player 2 does, the outcome maps to 1; and the goal of Player 2 is to ensure that the outcome maps to 0. A *quantitative objective* is a *value* function $f : \Omega \rightarrow \mathbb{R}$ and the goal of Player 1 is to maximize the value f of the play, whereas the goal of Player 2 is to minimize it. We only consider quantitative objectives with which map plays to values in $[0, 1]$. Given a boolean objective Φ, a play ρ is *winning* for Player 1 (Player 2) if $\Phi(\rho) = 1$ ($\Phi(\rho) = 0$). A strategy π is a *winning strategy* for Player p if every play conforming to π is winning for Player p.

For a quantitative objective f, the value of the game for Player 1 is defined as the supremum of the values of plays attainable against any Player 2 strategy, i.e., $\sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} f(out(\pi_1, \pi_2))$. The value of the game for Player 2 is defined analogously. A strategy is an *optimal strategy* for a player if it assures a outcome equal to her value of the game. Similarly, a strategy is an *ϵ-optimal strategy* for a maximizing (resp. minimizing) player if it assures an outcome that is no more that ϵ smaller (resp. larger) than the value of the game.

We consider ω-regular boolean objectives and the following quantitative objectives. Given a game graph with the weight function v and a play $\rho = \rho_0 \rho_1 \rho_2 \ldots$, for all $i \geq 0$, let $v_i = v((\rho_i, \sigma_i, \rho_{i+1}))$.

- $LimAvg(\rho) = \liminf_{n \rightarrow \infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} v_i$
- $Disc_\lambda(\rho) = \liminf_{n \rightarrow \infty} (1 - \lambda) \cdot \sum_{i=0}^{n-1} \lambda^i \cdot v_i$ where $0 < \lambda < 1$.

LimAvg is the long-run average of the weights occurring in a play, whereas $Disc_\lambda$ is the discounted sum of the weights. Therefore, *LimAvg* gives more importance to the infinite suffix of a play whereas $Disc_\lambda$ gives more importance to the finite prefix of a play.

Note that for *LimAvg* and *Disc* objectives, optimal memoryless strategies exist for both players [11,20]. Also, for qualitative objectives specified as Büchi conditions, memoryless winning strategies exist for both players, and for other ω-regular conditions, finite-memory winning strategies exist.

Also, consider the following family of objectives where a boolean ω-regular objective and a quantitative objective f are combined as follows. If a play ρ satisfies the boolean objective, then the value of ρ is the value according to f; otherwise, the value of the ρ is the maximum possible value of f (in our case, it is always 1). When $f = \mathit{LimAvg}$ and the ω-regular objective is a parity objective, ϵ-optimal finite-memory strategies exist [5]. This result can be extended to arbitrary ω-regular objectives as all ω-regular objectives can be expressed as parity objectives with the *latest appearance records* memory [13]. Such objectives are called ω-regular *LimAvg* objectives.

2.1 Qualitative Simulation Games

The simulation preorder [17] is a useful and polynomially computable relation to compare two transition systems. In [1] this relation was extended to alternating simulation between alternating transition systems. For systems with fairness conditions, the simulation relation was extended to fair simulation in [15]. These relations can be computed by solving games with boolean objectives.

Simulation and Alternating Simulation. Consider two transition systems $A = \langle S, \Sigma, E, s_0 \rangle$ and $A' = \langle S', \Sigma, E', s'_0 \rangle$. The system A' *simulates* the system A if there exists a relation $H \subseteq S \times S'$ such that (a) $(s_0, s'_0) \in H$; and (b) $\forall s, t \in S, s' \in S' : (s, s') \in H \wedge (s, \sigma, t) \in E \Rightarrow (\exists t' : (s', \sigma, t') \in E' \wedge (s', t') \in H)$.

For two alternating transition systems $A = \langle S, S_1, S_2, \Sigma, E, s_0 \rangle$ and $A' = \langle S', S'_1, S'_2, \Sigma, E', s'_0 \rangle$, *alternating simulation* of A by A' holds if there exists a relation $H \subseteq S \times S'$ such that $(s_0, s'_0) \in H$ and $\forall s \in S, s' \in S' : (s, s') \in H \Rightarrow (s \in S_1 \Leftrightarrow s' \in S'_1)$; and

- $\forall s \in S, s' \in S' : ((s, s') \in H \wedge s \in S_1) \Rightarrow \forall (s, \sigma, t) \in E : (\exists (s', \sigma, t') \in E' : (t, t') \in H)$.
- $\forall s \in S, s' \in S' : ((s, s') \in H \wedge s \in S_2) \Rightarrow \exists (s', \sigma, t') \in E' : (\forall (s, \sigma, t) \in E : (t, t') \in H)$.

Simulation and Alternating Simulation Games. Given two (alternating) transition systems, A and A', we can construct a game such that, (alternating) simulation of A by A' holds if and only if Player 2 has a winning strategy in the game

Given two weighted transition systems A and A' with the same alphabet, we define the corresponding *quantitative simulation game graph* $G_{A,A'}$ as $\langle S \times (\Sigma \cup \{\#\}) \times S' \cup \{s_{\mathrm{err}}\}, S_1^G, S_2^G, \Sigma, E^G, (s_0, \#, s'_0) \rangle$ where $S_1^G = (S \times \{\#\} \times S') \cup \{s_{\mathrm{err}}\}$ and $S_2^G = (S \times \Sigma \times S')$. Each transition of the game graph corresponds to a transition in either A or A' as follows:

- $((s, \#, s'), \sigma, (t, \sigma, s')) \in E^G \Leftrightarrow (s, \sigma, t) \in E$
- $((s, \sigma, s'), \sigma, (s, \#, t')) \in E^G \Leftrightarrow (s', \sigma, t') \in E'$

For each of the above transitions, the weight is the same as the weight of the corresponding transition in A or A'. If there is no outgoing transition from a particular state, transitions to s_{err} are added with all symbols. The state s_{err} is a sink with transitions to itself on all symbols. Each of these transitions has weight 1 (the maximum possible value of a quantitative objective).

For classical simulation games, we consider the same game graph without weights. The objective for Player 2 is to reach s_{err} and for Player 1 to avoid it. Intuitively, in every state, Player 1 chooses a transition of A and Player 2 has to match it by picking a transition of A'. If Player 2 cannot match at some point, Player 1 wins that play. It is easy to see that A' simulates A iff there is a winning strategy for Player 2 in this game.

We can extend the simulation game to an alternating simulation game. We informally define the *quantitative alternating simulation game graph*. The formal definition can be found in the companion report [3]. Given two quantitative alternating transition systems A and A' , we define the quantitative alternating simulation game graph $H_{A,A'}$ as follows. If A is at state s and $s \in S_1$, Player 1 chooses a transition of A and Player 2 has to match it with a transition of A'; and if A is at s and $s \in S_2$, Player 2 has to choose a transition of A' and Player 1 has to choose a transition of A to match it. If there cannot be a match, the control moves to the error state s_{err}. As before, the transitions have the same weight as in the individual systems.

We consider the game graph without weights to define the alternating simulation game $\mathcal{H}^{A,A'}$ and the objective of the Player 1 is to ensure that the play reaches s_{err}. It can be seen that alternating simulation holds iff there exists a winning strategy for Player 2 .

Fair Simulation. Given two (alternating) transitions systems with fairness conditions A^F and $A'^{F'}$, the fair simulation game is played in the same game graph $G_{A,A'}$ ($H_{A,A'}$) as the simulation game. However, in addition to matching the symbol in each step, Player 2 has to ensure that if the sequence of transitions of A chosen by Player 1 satisfies the fairness condition F, then the sequence of A' transitions chosen satisfy the fairness condition F'.

2.2 Quantitative Simulation Games

We define a generalized notion of simulation games called quantitative simulation games where the simulation objectives are replaced by quantitative objectives.

Quantitative Simulation Games. Given two quantitative (alternating) transition systems A and A', and $f \in \{LimAvg, Disc_\lambda\}$, the *quantitative (alternating) simulation game* is played on the quantitative (alternating) simulation game graph $G_{A,A'}$ ($H_{A,A'}$) with the objective of Player 1 being to maximize the f value of the play. We denote this game as $\mathcal{Q}^f_{A,A'}$ ($\mathcal{P}^f_{A,A'}$).

Quantitative Fair Simulation Games. Analogous to quantitative (alternating) simulation games, the fair versions between two transition systems with fairness conditions. The quantitative objective for this game is the ω-regular

LimAvg objective which is the combination of *LimAvg* objective and the boolean fair (alternating) simulation game objective.

We do not use $f = Disc_\lambda$ along with fairness conditions as the two objectives are independent. The $Disc_\lambda$ objectives mainly consider the finite prefix of a play, whereas fairness conditions consider only the infinite suffix.

2.3 Modification Schemes

We will use quantitative simulation games to measure various properties of systems. For computing these properties, we need to use small modifications of the original systems. For example, when trying to compute the distance as the number of errors an implementation commits, we add to the specification some error recovery behavior. However, we impose strict rules on these modifications to ensure that the modified system retains the structure of the original system.

A *modification scheme* is a function m from transition systems to quantitative (alternating) transition systems, which can be computed using the following steps: (a) Edges may be added to the transition system and each state may be replaced by a local subgraph. All edges of the graph have to be preserved; (b) Every edge of the system is associated with a weight from $\mathbb{Q}$. We present two examples of modification schemes.

Output Modification. This scheme is used to add behavior to a system that allows it to output an arbitrary symbol while moving to a state specified by an already existing transition. For every transition (s, σ, s'), transitions with different symbols are added to the system i.e., $\{(s, \alpha, s') \mid \alpha \in \Sigma\}$. These transitions are given a weight of 2 to prohibit their free use. All other transitions have the weight zero. Given a system T, we denote the modified system as $OutMod(T)$.

Error Modification. In a perfectly functioning system, errors may occur due to unpredictable events. We model this with an alternating transition system with one player modeling the original system (Player 1) and the other modeling the controlled error (Player 2). At every state, Player 2 chooses whether or not a error occurs by choosing one of the two successors. From one of these states, Player 1 can choose the original successors of the state and from the other, she can choose either one of the original successors or one of the error transitions. We penalize Player 2 for the choice of not allowing errors to happen.

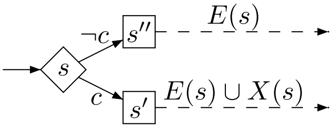


Fig. 2. Graph for *ErrMod*

Given $T = \langle S, \Sigma, E, s_0 \rangle$ we define $ErrMod(T)$ to be the quantitative alternating transition system obtained by replacing each state s by the graph in Figure 2. If an error is allowed (modeled by the c edge), then all transitions that differ from original transitions only in the symbol are added (represented by $X(s)$ in Figure 2). Only the transitions labeled $\neg c$ are given the weight 2. The rest are given the weight 0. The system $ErrMod_\emptyset(T)$ denotes a system where

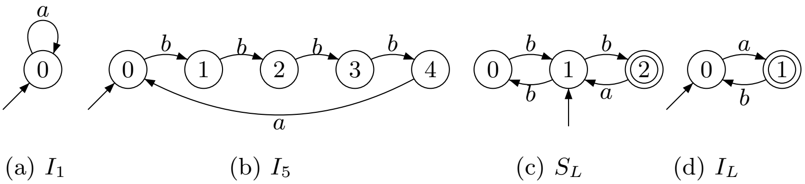


Fig. 3. Example Systems

no additional transitions where introduced, only the states were replaced by a subgraph from Figure 2 (with X being the empty set).

In addition to the above schemes, we define the trivial modification scheme *NoMod* where no changes are made except to give every edge the weight 0.

3 Simulation Distances

Correctness. Given a specification T_2 and an implementation T_1, such that T_1 is incorrect with respect to T_2, the correctness distance measures the degree of "incorrectness" of T_1. Even a single nonconformant behavior can destroy the boolean simulation relation. Here we present a game which is not as strict and measures the minimal number of required errors, i.e. the minimal number of times the specification has to use nonmatching symbols when simulating the implementation.

Definition 3.1 (Correctness distance). *Let $f = LimAvg$ or $f = Disc_\lambda$. The* correctness distance $d^f_{cor}(T_1, T_2)$ *from system T_1 to system T_2 is the Player 1 value of the quantitative simulation game $\mathcal{C}^f_{T_1,T_2} = \mathcal{Q}^f_{NoMod(T_1),OutMod(T_2)}$.*

The game $\mathcal{C}$ can be intuitively understood as follows. Given two systems T_1 and T_2, we are trying to simulate the system T_1 by T_2, but the specification T_2 is allowed to make errors, to "cheat", but she has to pay a price for such a choice. As the simulating player is trying to minimize the value of the game, she is motivated not to cheat. The value of the game can thus be seen as measuring how often she can be forced to cheat, that is, how often on average the implementation commits an error. If the implementation is correct (T_2 simulates T_1), then the correctness distance is 0. The value of the game is either the *LimAvg* or the *Disc* of the number of errors. If the objective f is *LimAvg*, then the value is the long run average of the errors, whereas if the objective f is $Disc_\lambda$, the errors which occur earlier are given more importance and the value is the discounted sum of the position of the errors. Therefore, the *Disc* and *LimAvg* games are concerned with prefixes and infinite suffixes of the behaviors respectively.

We present a few example systems and their distances here to demonstrate the fact that the above game measures distances that correspond to intuition. In Figure 3 and Figure 1, S_1 is the specification system against which we want to measure the systems I_1 through I_5. In this case, the specification says that

Table 1. Distances according to the Correctness, Coverage and Robustness game

T_1	T_2	$d_{\text{cor}}^{LimAvg}(T_1, T_2)$	$d_{\text{cov}}^{LimAvg}(T_1, T_2)$	$d_{\text{rob}}^{LimAvg}(T_1, T_2)$
S_1	S_1	0	0	1
S_1	I_1	0	2/3	1/3
S_1	I_2	0	1/3	2/3
S_1	I_3	1/3	1	1
S_1	I_4	1/4	3/4	1
S_1	I_5	1/5	4/5	1
S_L	I_L	1/2	1	1

there cannot be more than two b's in a row. Also, we have a specification with a liveness condition S_L against which we want to measure the implementation I_L. The distances between these systems according to the *LimAvg* correctness game are summarized in Table 1.

Among the systems which do not satisfy the specification S_1, i.e. I_3, I_4 and I_5, we showed in the introduction that the distance from I_3 to S_1 is 1/3, while the distance from I_4 to S_1 is 1/4. However, surprisingly the distance from I_5 to S_1 is less than the distance from I_4. In fact, the distances reflect on the long run the number of times the specification has to err to simulate the implementation.

In case of the specification S_L and implementation I_L with liveness conditions, the specification can take the left branch to state 0 to get a penalty of $\frac{1}{2}$ or take the right branch to state 2 to get a penalty of 1. However, it needs to take the right branch infinitely often to satisfy the liveness condition. To achieve the distance of $\frac{1}{2}$, the specification needs infinite memory so that it can take the right branch lesser and lesser number of times. In fact, if the specification has a strategy with finite-memory of size m, it can achieve a distance of $\frac{1}{2} + \frac{1}{2m}$.

Coverage. We present the dual game of the one presented above. Here, we measure the behaviors that are present in one system but not in the other system. Given a specification T_2 and an implementation T_1, the coverage distance corresponds to the behavior of the specification which is farthest from any behaviour of the implementation. Hence, we have that the coverage distance from a system T_1 to a system T_2 is the correctness distance from T_2 to T_1.

Definition 3.2 (Coverage distance). *Let $f = LimAvg$ or $f = Disc_\lambda$. The* coverage distance $d_{cov}^f(T_1, T_2)$ *from system T_1 to system T_2 is the Player 1 value of the quantitative simulation game $\mathcal{V}_{T_1,T_2}^f = \mathcal{Q}_{NoMod(T_2),OutMod(T_1)}^f$.*

$\mathcal{V}$ measures the distance from T_1 to T_2 as the minimal number of errors that have to be committed by T_1 to cover all the behaviors of T_2. We present examples of systems and their distances according to $\mathcal{V}^{LimAvg}$. We use the example systems in Figures 3 and 1. The distances are summarized in Table 1.

Robustness. Given a specification system and a correct implementation, the notion of robustness presented here is a measure of the number of errors by the implementation that makes it nonconformant to the specification. The more such errors tolerated by the specification, the more robust the implementation. In other words, the distance measures the number of critical points, or points where an error will lead to an unacceptable behavior. The lower the value of the robustness distance, the more robust an implementation is. In case of an incorrect implementation, the simulation of the implementation does not hold irrespective of implementation errors. Hence, in that case, the robustness distance will be 1.

Definition 3.3 (Robustness distance). *Let* $f = LimAvg$ *or* $f = Disc_\lambda$. *The* robustness distance $d^f_{rob}(T_1, T_2)$ *from system* T_1 *to system* T_2 *is the Player 1 value of the quantitative alternating simulation game* $\mathcal{R}^f_{T_1,T_2} = \mathcal{P}^f_{ErrMod(T_1),ErrMod_\emptyset(T_2)}$.

The game $\mathcal{R}_{ErrMod(T_1),ErrMod_\emptyset(T_2)}$ is played in the following steps: (a) The specification T_2 chooses whether the implementation T_1 is allowed to make an error; (b) The implementation chooses a transition on the implementation system. It is allowed to err based on the specification choice in the previous step; and (c) Specification chooses a matching move to simulate the implementation.

The specification tries to minimize the number of moves where it prohibits implementation errors (without destroying the simulation relation), whereas the implementation tries to maximize it. Intuitively, the positions where the specification cannot allow errors are the critical points for the implementation.

In the game played between S_1 and S_1, every position is critical. At each position, if an error is allowed, the system can output three b's in a row by using the error transition to return to state 0 while outputting a b. The next two moves can be b's irrespective whether errors are allowed or not. This breaks the simulation. Now, consider I_1. This system can be allowed to err every two out of three times without violating the specification. This shows that I_1 is more robust than S_1 for implementing S_1. The list of distances is summarized in Table 1.

Computation of Simulation Distances. The computational complexity of computing the three distances defined here is the same as solving the value problem for the respective games.

For systems without fairness conditions, the d_{cor}, d_{cov} and d_{rob} games are simple graph games with *LimAvg* or $Disc_\lambda$ objectives. The decision problem for these games is in NP $\cap$ co-NP [20], but no PTIME algorithm is known. However, for *LimAvg* objectives the existence of an algorithm polynomial in unary encoded weights implies that the computation of the distances can be achieved in polynomial time as we use constant weights. Using the algorithm of [20], in the case without fairness conditions d_{cor}, d_{cov} and d_{rob} distances can be computed in time $O((|S||S'|)^3 \cdot (|E||S'| + |E'||S|))$ where S and S' are state spaces of the two transition systems; and E and E' are the sets of transitions of the two systems. A variation of the algorithm in [20] gives a PTIME algorithm for the $Disc_\lambda$ objectives (given a fixed λ).

For systems with Büchi (weak fairness) conditions, the corresponding games are graph games with *LimAvg* parity games, for which the decision problem is in NP $\cap$ co-NP. However, the use of constant weights and the fact that the implication of two Büchi conditions can be expressed as a parity condition with no more than 3 priorities leads to a polynomial algorithm. Using the algorithm presented in [5], we get a $O((|S||S'|)^3 \cdot (|E||S'| + |E'||S|))$ algorithm.

For systems with Streett (strong fairness) conditions, the corresponding games are graph games with *LimAvg* ω-regular conditions. For an ω-regular *LimAvg* game of n states, we can use the latest appearance records to convert into an equivalent parity game of $2^{O(n \log(n))}$ states and edges; and n priorities. The algorithm of [5] gives a $2^{O(n \log(n))}$ algorithm where $n = |S| \cdot |S'|$.

4 Properties of Simulation Distances

We present quantitative analogues of boolean properties of the simulation preorders. Proofs omitted are included in the companion report [3].

Triangle Inequality. Classical simulation relations satisfy the reflexivity and transitivity property which makes them preorders. In an analogous way, we show that the correctness and coverage distances satisfy the quantitative reflexivity and the triangle inequality properties. This makes them directed metrics [8].

Theorem 1. *d^f_{cor} is a directed metric for $f \in \{LimAvg, Disc_\lambda\}$, i.e.:*
- $\forall S \in \mathcal{S} : d^f_{cor}(S, S) = 0$
- $\forall S_1, S_2, S_3 \in \mathcal{S} : d^f_{cor}(S_1, S_3) \leq d^f_{cor}(S_1, S_2) + d^f_{cor}(S_2, S_3)$

Proof: We will prove the result for systems with fairness conditions. The case without fairness conditions is analogous. Consider any $\epsilon > 0$. Let τ_2 and τ_3 be $\frac{\epsilon}{2}$-optimal finite strategies for Player 2 in $\mathcal{C}_{S_1,S_2}$ and $\mathcal{C}_{S_2,S_3}$ respectively. Now, we construct a finite-memory strategy τ^* for Player 2 in $\mathcal{C}_{S_1,S_3}$. If M_2 and M_3 are the memories of τ_2 and τ_3 respectively, the memory of τ^* will be $M_2 \times S_2 \times M_3$. The strategy τ^* works as follows. Let the state of the game be $(s_1, \#, s_3)$ and the memory of τ^* be (m_2, s_2, m_3).
- Let Player 1 choose to move according to the S_1 transition (s_1, σ_1, s_1') to the game state (s_1', σ_1, s_3). Consider the game position (s_1', σ_1, s_2) in $\mathcal{C}_{S_1,S_2}$ and let the τ_2 memory be at state m_2. Say τ_2 updates its memory to m_2' and chooses the successor $(s_1', \#, s_2')$ with transition symbol σ_1. Let the corresponding *OutMod*(S_2) transition be (s_2, σ_1, s_2').
- If the transition (s_2, σ_1, s_2') exists in S_2, then let $\sigma_2' = \sigma_1$. Otherwise, there will exist (s_2, σ_2, s_2') in S_2 for some σ_2. Let $\sigma_2' = \sigma_2$. Now, consider the game position (s_2', σ_2', s_3) in $\mathcal{C}_{S_2,S_3}$ and the memory state m_3 of τ_3. Say τ_3 updates its memory to m_3' and chooses the successor $(s_2', \#, s_3')$ and the transition symbol σ_2'. Let the corresponding *OutMod*(S_3) transition be (s_3, σ_2', s_3').
- The memory of τ^* is updated to (m_2', s_2', m_3') and τ^* chooses the successor $(s_1', \#, s_3')$ with the transition symbol σ_1. The corresponding transition (s_3, σ_1, s_3') exists in *OutMod*(S_3) as there exists a transition with the same source and destination as (s_3, σ_2', s_3').

$$\left.\begin{array}{l}\left.\begin{array}{l} s_{1,0} \xrightarrow{\sigma_1(v_{1,0})} s_{1,1} \xrightarrow{\sigma_1(v_{1,1})} s_{1,2}\cdots \\ s_{2,0} \xrightarrow{\sigma_1(v_{2,0})} s_{2,1} \xrightarrow{\sigma_1(v_{2,1})} s_{2,2}\cdots \end{array}\right\}\rho_1 \\ \left.\begin{array}{l} s_{2,0} \xrightarrow{\sigma_2'(v_{2,0})} s_{2,1} \xrightarrow{\sigma_2'(v_{2,1})} s_{2,2}\cdots \\ s_{3,0} \xrightarrow{\sigma_1(v_{3,0})} s_{3,1} \xrightarrow{\sigma_1(v_{3,1})} s_{3,2}\cdots \end{array}\right\}\rho_2 \end{array}\right\}\rho$$

If Player 2 cannot match σ_1 with a zero weight transition while playing according to τ^*, either τ_2 or τ_3 would have also taken a non-zero weight transition. Using this fact, we can easily prove the required property. Fix an arbitrary finite-memory Player 1 strategy σ. Now, let the play proceed according to the strategy τ^*. From the moves of the game and the state of the memory of τ^*, we can extract four transitions for each round of play as above, i.e. an S_1 transition (s_1, σ_1, s_1'), an *OutMod*(S_2) transition (s_2, σ_1, s_2'), an S_2 transition (s_2, σ_2', s_2') and an *OutMod*(S_3) transition (s_3, σ_1, s_3'). We depict the situation in the above figure.

The play ρ in $\mathcal{C}_{S_1,S_3}$ corresponds to the transitions in the first and the last rows. This play can be decomposed into plays ρ_1 and ρ_2 in $\mathcal{C}_{S_1,S_2}$ and $\mathcal{C}_{S_2,S_3}$ by taking only the transitions in the first two and last two rows respectively. Now, by the observation in the previous paragraph, each move in ρ has weight 2 only if one of the corresponding moves in ρ_1 or ρ_2 have weight 2. Let us denote the n^{th} move in a play η by η^n. If both S_1 and S_3 sequence of moves in ρ are fair or if S_1 sequence is unfair, we have the following for the *LimAvg* case.

$$\begin{aligned}
\nu(\rho) &= \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho^i) \leq \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} \left(v(\rho_1^i) + v(\rho_2^i)\right) \\
&= \lim_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} \left(v(\rho_1^i) + v(\rho_2^i)\right) = \lim_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho_1^i) + \lim_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho_2^i) \\
&= \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho_1^i) + \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho_2^i) \\
&\leq d_{\mathrm{cor}}(S_1, S_2) + \frac{\epsilon}{2} + d_{\mathrm{cor}}(S_2, S_3) + \frac{\epsilon}{2} = d_{\mathrm{cor}}(S_1, S_2) + d_{\mathrm{cor}}(S_2, S_3) + \epsilon
\end{aligned}$$

All the strategies we are considering are finite-memory, and hence, each sequence of weights is ultimately repeating. Therefore, we can use lim and lim inf interchangeably in the above equations. The case for *Disc*$_\lambda$ is much simpler and not shown here.

Hence, we have that the value of the play satisfies the required inequality for the case that both S_1 and S_3 perform fair computations. In the case that S_1 sequence is fair and S_3 sequence is not fair, the value of the play will be 1. However, by construction the value of either ρ_1 or ρ_2 will also be 1 and hence the inequality holds.

Therefore, given an ϵ, we have strategy for Player 2 which assures a value less than $d_{\mathrm{cor}}(S_1, S_2) + d_{\mathrm{cor}}(S_2, S_3) + \epsilon$ for both the *LimAvg* and *Disc*$_\lambda$ case. Hence, we have the required triangle inequality.

It can be shown by construction of a Player 2 strategy that copies every Player 1 move that $d_{\mathrm{cor}}(S, S) = 0$. Hence, we have the result. ■

Theorem 2. *d^f_{cov} is a directed metric when $f \in \{LimAvg, Disc_\lambda\}$, i.e. :*

- $\forall S \in \mathcal{S} : d^f_{cov}(S, S) = 0$
- $\forall S_1, S_2, S_3 \in \mathcal{S} : d^f_{cov}(S_1, S_3) \leq d^f_{cov}(S_1, S_2) + d^f_{cov}(S_2, S_3)$

The robustness distance satisfies the triangle inequality, but not the quantitative reflexivity. The system S_1 in Figure 1 is a witness system that violates $d_{\text{rob}}(S_1, S_1) = 0$. In fact, for *LimAvg* objectives and any rational value $v \in [0, 1]$, it is easy to construct a system S_v such that $d_{\text{rob}}(S_v, S_v) = v$.

Theorem 3. *d^f_{rob} conforms to the triangle inequality for $f \in \{LimAvg, Disc_\lambda\}$, i.e. : $\forall S_1, S_2, S_3 \in \mathcal{S} : d^f_{rob}(S_1, S_3) \leq d^f_{rob}(S_1, S_2) + d^f_{rob}(S_2, S_3)$*

Compositionality In the qualitative case, compositionality theorems help analyse large systems by decomposing them into smaller components. For example, simulation is preserved when components are composed together. We show that in the quantitative case, the distance between the composed systems is bounded by the sum of the distances between individual systems.

If A and A' are two transition systems, we define *asynchronous and synchronous composition* of the two systems, written as $A \parallel A'$ and $A \times A'$ respectively as follows: (a) The state space is $S \times S'$; (b) $((s, s'), \sigma, (t, t'))$ is a transition of $A \parallel A'$ iff (s, σ, t) is a transition of A and $s' = t'$ or (s', σ, t') is a transition of A' and $s = t$, and (c) $((s, s'), \sigma, (t, t'))$ is a transition of $A \times A'$ iff (s, σ, t) is a transition of A and (s', σ, t') is a transition of A'.

The following theorems show that the simulation distances between whole systems is bounded by the sum of distances between the individual components.

Theorem 4. *The correctness, coverage and robustness distances satisfy the following property, when $f \in \{LimAvg, Disc_\lambda\}$:*

$$d^f(S_1 \times S_2, T_1 \times T_2) \leq d^f(S_1, T_1) + d^f(S_2, T_2)$$

Theorem 5. *The correctness, coverage and robustness distances satisfy the following property when $f = LimAvg$.*

$$d^f(S_1 \parallel S_2, T_1 \parallel T_2) \leq \alpha.d^f(S_1, T_1) + (1 - \alpha).d^f(S_2, T_2)$$

where α is the fraction of times S_1 is scheduled in $S_1 \parallel S_2$ in the long run, assuming that the fraction has a limit in the long run.

Existential and Universal Abstraction. Classically, properties of systems are studied by studying the properties of over-approximations and under-approximations. In an analogous way, we prove that the distances between systems is bounded from above and below by distances between abstractions of the systems. Given $T = \langle S, \Sigma, E, s_0 \rangle$, an existential (universal) abstraction of it is a system whose states are disjoint subsets of S and an edge exists between two classes iff there exists an edge between one pair (all pairs) of states in the classes.

Theorem 6. *Consider a specification S and an implementation I. Let $S^\exists$ and $I^\exists$ be existential abstractions, and $S^\forall$ and $I^\forall$ be universal abstractions of S and I respectively. The correctness, coverage and robustness distances satisfy the three following properties when $f \in \{LimAvg, Disc_\lambda\}$:*

(a) $d^f_{cor}(I^\forall, S^\exists) \leq d^f_{cor}(I, S) \leq d^f_{cor}(I^\exists, S^\forall)$
(b) $d^f_{cov}(I^\exists, S^\forall) \leq d^f_{cov}(I, S) \leq d^f_{cov}(I^\forall, S^\exists)$
(c) $d^f_{rob}(I^\forall, S^\exists) \leq d^f_{rob}(I, S) \leq d^f_{rob}(I^\exists, S^\forall)$

5 Robustness of Forward Error Correction Systems

Forward Error Correction systems (FECS) are a mechanism of error control for data transmission on noisy channels. A very important characteristic of these error correction systems is the *maximum tolerable bit-error rate*, which is the maximum number of errors the system can tolerate while still being able to successfully decode the message. We show that this property can be measured as the d_{rob} distance between a system and an ideal system (specification).

Table 2. FECS' robustness

T_1	T_2	$d_{\text{rob}}(T_1, T_2)$
None	Ideal	1
Hamming	Ideal	6/7
TMR	Ideal	2/3

We will examine three forward error correction systems: one with no error correction facilities, the Hamming(7,4) code [14], and triple modular redundancy (TMR) that by design can tolerate no errors, one error in seven and three bits respectively. We measure the robustness with respect to an ideal system which can tolerate an unbounded number of errors. For the pseudo-code for the three systems we are examining, the user is referred to the companion report [3]. The only errors we allow are bit flips during transmission.

These systems were modelled and the values of d_{rob} of these systems measured against the ideal system are summarized in Table 2. The robustness values mirror the error tolerance values. In fact, each robustness value is equal to $1 - e$ where e is the corresponding error tolerance value.

References

1. 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)
2. Caspi, P., Benveniste, A.: Toward an approximation theory for computerised control. In: Sangiovanni-Vincentelli, A.L., Sifakis, J. (eds.) EMSOFT 2002. LNCS, vol. 2491, pp. 294–304. Springer, Heidelberg (2002)
3. Černý, P., Henzinger, T.A., Radhakrishna, A.: Simulation distances. Technical Report IST-2010-0003, IST Austria (June 2010)
4. Chatterjee, K., Doyen, L., Henzinger, T.: Quantitative languages. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 385–400. Springer, Heidelberg (2008)
5. Chatterjee, K., Henzinger, T.A., Jurdzinski, M.: Mean-payoff parity games. In: LICS, pp. 178–187 (2005)
6. de Alfaro, L., Faella, M., Stoelinga, M.: Linear and branching system metrics. IEEE Trans. Software Eng. 35(2), 258–273 (2009)
7. de Alfaro, L., Henzinger, T., Majumdar, R.: Discounting the future in systems theory. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 1022–1037. Springer, Heidelberg (2003)

8. de Alfaro, L., Majumdar, R., Raman, V., Stoelinga, M.: Game refinement relations and metrics. Logical Methods in Computer Science 4(3) (2008)
9. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Metrics for labelled Markov processes. Theor. Comput. Sci. 318(3), 323–354 (2004)
10. Droste, M., Gastin, P.: Weighted automata and weighted logics. Theor. Comput. Sci. 380(1-2), 69–86 (2007)
11. Ehrenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. International Journal of Game Theory, 163–178 (1979)
12. Fenton, N.: Software Metrics: A Rigorous and Practical Approach, Revised (Paperback). Course Technology (1998)
13. Gurevich, Y., Harrington, L.: Trees, automata, and games. In: STOC, pp. 60–65 (1982)
14. Hamming, R.W.: Error detecting and error correcting codes. Bell System Tech. J. 29, 147–160 (1950)
15. Henzinger, T.A., Kupferman, O., Rajamani, S.K.: Fair simulation. Information and Computation, 273–287 (1997)
16. Lincke, R., Lundberg, J., Löwe, W.: Comparing software metrics tools. In: ISSTA, pp. 131–142 (2008)
17. Milner, R.: An algebraic definition of simulation between programs. In: IJCAI, pp. 481–489 (1971)
18. van Breugel, F.: An introduction to metric semantics: operational and denotational models for programming and specification languages. Theor. Comput. Sci. 258(1-2), 1–98 (2001)
19. van Breugel, F., Worrell, J.: Approximating and computing behavioural distances in probabilistic transition systems. Theo. Comp. Sci. 360(1-3), 373–385 (2006)
20. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theor. Comput. Sci. 158(1&2), 343–359 (1996)

Mean-Payoff Automaton Expressions*

Krishnendu Chatterjee[1], Laurent Doyen[2], Herbert Edelsbrunner[1],
Thomas A. Henzinger[1,3], and Philippe Rannou[2,3,4]

[1] IST Austria (Institute of Science and Technology Austria)
[2] LSV, ENS Cachan & CNRS, France
[3] EPFL Lausanne, Switzerland
[4] ENS Cachan Bretagne, Rennes, France

Abstract. Quantitative languages are an extension of boolean languages that assign to each word a real number. Mean-payoff automata are finite automata with numerical weights on transitions that assign to each infinite path the long-run average of the transition weights. When the mode of branching of the automaton is deterministic, nondeterministic, or alternating, the corresponding class of quantitative languages is not *robust* as it is not closed under the pointwise operations of max, min, sum, and numerical complement. Nondeterministic and alternating mean-payoff automata are not *decidable* either, as the quantitative generalization of the problems of universality and language inclusion is undecidable.

We introduce a new class of quantitative languages, defined by *mean-payoff automaton expressions*, which is robust and decidable: it is closed under the four pointwise operations, and we show that all decision problems are decidable for this class. Mean-payoff automaton expressions subsume deterministic mean-payoff automata, and we show that they have expressive power incomparable to nondeterministic and alternating mean-payoff automata. We also present for the first time an algorithm to compute distance between two quantitative languages, and in our case the quantitative languages are given as mean-payoff automaton expressions.

1 Introduction

Quantitative languages L are a natural generalization of boolean languages that assign to every word w a real number $L(w) \in \mathbb{R}$ instead of a boolean value. For instance, the value of a word (or behavior) can be interpreted as the amount of some resource (e.g., memory consumption, or power consumption) needed to produce it, or bound the long-run average available use of the resource. Thus quantitative languages can specify properties related to resource-constrained programs, and an implementation L_A satisfies (or refines) a specification L_B if $L_A(w) \leq L_B(w)$ for all words w. This notion of refinement is a *quantitative generalization of language inclusion*, and it can be used to check for example if for each behavior, the long-run average response time of the system lies below the specified average response requirement. Hence it is crucial to identify

* This research was supported by EPFL, IST Austria, LSV@ENS Cachan & CNRS, and the following grants: the European Union project COMBEST, the European Network of Excellence ArtistDesign, the DARPA grant HR0011-05-1-0057, and the NSF grant DBI-0820624.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 269–283, 2010.

some relevant class of quantitative languages for which this question is decidable. The other classical decision questions such as emptiness, universality, and language equivalence have also a natural quantitative extension. For example, the *quantitative emptiness problem* asks, given a quantitative language L and a threshold $\nu \in \mathbb{Q}$, whether there exists some word w such that $L(w) \geq \nu$, and the *quantitative universality problem* asks whether $L(w) \geq \nu$ for all words w. Note that universality is a special case of language inclusion (where $L_A(w) = \nu$ is constant).

Weighted *mean-payoff automata* present a nice framework to express such quantitative properties [4]. A weighted mean-payoff automaton is a finite automaton with numerical weights on transitions. The value of a word w is the maximal value of all runs over w (if the automaton is nondeterministic, then there may be many runs over w), and the value of a run r is the long-run average of the weights that appear along r. A mean-payoff extension to alternating automata has been studied in [5]. Deterministic, nondeterministic and alternating mean-payoff automata are three classes of mean-payoff automata with increasing expressive power. However, none of these classes is closed under the four pointwise operations of max, min (which generalize union and intersection respectively), numerical complement[1], and sum (see Table 1). Deterministic mean-payoff automata are not closed under max, min, and sum [6]; nondeterministic mean-payoff automata are not closed under min, sum and complement [6]; and alternating mean-payoff automata are not closed under sum [5]. Hence none of the above classes is *robust* with respect to closure properties.

Moreover, while deterministic mean-payoff automata enjoy decidability of all quantitative decision problems [4], the quantitative language-inclusion problem is undecidable for nondeterministic and alternating mean-payoff automata [10], and thus also all decision problems are undecidable for alternating mean-payoff automata. Hence although mean-payoff automata provide a nice framework to express quantitative properties, there is no known class which is both robust and decidable (see Table 1).

In this paper, we introduce a new class of quantitative languages that are defined by *mean-payoff automaton expressions*. An expression is either a deterministic mean-payoff automaton, or it is the max, min, or sum of two mean-payoff automaton expressions. Since deterministic mean-payoff automata are closed under complement, mean-payoff automaton expressions form a robust class that is closed under max, min, sum and complement. We show that (a) all decision problems (quantitative emptiness, universality, inclusion, and equivalence) are decidable for mean-payoff automaton expressions; (b) mean-payoff automaton expressions are incomparable in expressive power with both the nondeterministic and alternating mean-payoff automata (i.e., there are quantitative languages expressible by mean-payoff automaton expressions that are not expressible by alternating mean-payoff automata, and there are quantitative languages expressible by nondeterministic mean-payoff automata that are not expressible by mean-payoff automata expressions); and (c) the properties of cut-point languages (i.e., the sets of words with value above a certain threshold) for deterministic automata carry over to mean-payoff automaton expressions, mainly the cut-point language is ω-regular when the threshold is isolated (i.e., some neighborhood around the threshold contains no word). Moreover, mean-payoff automaton expressions can express all

[1] The numerical complement of a quantitative languages L is $-L$.

Table 1. Closure properties and decidability of the various classes of mean-payoff automata. Mean-payoff automaton expressions enjoy fully positive closure and decidability properties.

	Closure properties				Decision problems			
	max	min	sum	comp.	empt.	univ.	incl.	equiv.
Deterministic	×	×	×	✓[2]	✓	✓	✓	✓
Nondeterministic	✓	×	×	×	✓	×	×	×
Alternating	✓	✓	×	✓[2]	×	×	×	×
Expressions	✓	✓	✓	✓	✓	✓	✓	✓

examples in the literature of quantitative properties using mean-payoff measure [1,6,7]. Along with the quantitative generalization of the classical decision problems, we also consider the notion of *distance* between two quantitative languages L_A and L_B, defined as $\sup_w |L_A(w) - L_B(w)|$. When quantitative language inclusion does not hold between an implementation L_A and a specification L_B, the distance is a relevant information to evaluate how far they are from each other, as we may accept implementations that overspend the resource but we would prefer the least expensive ones. We present the first algorithm to compute the distance between two quantitative languages: we show that the distance can be computed for mean-payoff automaton expressions.

Our approach to show decidability of mean-payoff automaton expressions relies on the characterization and algorithmic computation of the *value set* $\{L_E(w) \mid w \in \Sigma^\omega\}$ of an expression E, i.e. the set of all values of words according to E. The value set can be viewed as an abstract representation of the quantitative language L_E, and we show that all decision problems, cut-point language and distance computation can be solved efficiently once we have this set.

First, we present a precise characterization of the value set for quantitative languages defined by mean-payoff automaton expressions. In particular, we show that it is not sufficient to construct the convex hull $\mathsf{conv}(S_E)$ of the set S_E of the values of simple cycles in the mean-payoff automata occurring in E, but we need essentially to apply an operator $F_{\min}(\cdot)$ which given a set $Z \subseteq \mathbb{R}^n$ computes the set of points $y \in \mathbb{R}^n$ that can be obtained by taking pointwise minimum of each coordinate of points of a set $X \subseteq Z$. We show that while we need to compute the set $V_E = F_{\min}(\mathsf{conv}(S_E))$ to obtain the value set, and while this set is always convex, it is not always the case that $F_{\min}(\mathsf{conv}(S_E)) = \mathsf{conv}(F_{\min}(S_E))$ (which would immediately give an algorithm to compute V_E). This may appear counter-intuitive because the equality holds in $\mathbb{R}^2$ but we show that the equality does not hold in $\mathbb{R}^3$ (Example 2).

Second, we provide algorithmic solutions to compute $F_{\min}(\mathsf{conv}(S))$, for a finite set S. We first present a constructive procedure that given S constructs a finite set of points S' such that $\mathsf{conv}(S') = F_{\min}(\mathsf{conv}(S))$. The explicit construction presents interesting properties about the set $F_{\min}(\mathsf{conv}(S))$, however the procedure itself is computationally expensive. We then present an elegant and geometric construction of $F_{\min}(\mathsf{conv}(S))$ as a set of linear constraints. The computation of $F_{\min}(\mathsf{conv}(S))$ is a new problem in

[2] Closure under complementation holds because LimInfAvg-automata and LimSupAvg-automata are dual. It would not hold if only LimInfAvg-automata (or only LimSupAvg-automata) were allowed.

computational geometry and the solutions we present could be of independent interest. Using the algorithm to compute $F_{\min}(\mathsf{conv}(S))$, we show that all decision problems for mean-payoff automaton expressions are decidable. Due to lack of space, most proofs are given in the fuller version [3].

Related works. Quantitative languages have been first studied over finite words in the context of probabilistic automata [17] and weighted automata [18]. Several works have generalized the theory of weighted automata to infinite words (see [14,12,16,2] and [13] for a survey), but none of those have considered mean-payoff conditions. Examples where the mean-payoff measure has been used to specify long-run behaviours of systems can be found in game theory [15,20] and in Markov decision processes [8]. The mean-payoff automata as a specification language have been investigated in [4,6,5], and extended in [1] to construct a new class of (non-quantitative) languages of infinite words (the multi-threshold mean-payoff languages), obtained by applying a query to a mean-payoff language, and for which emptiness is decidable. It turns out that a richer language of queries can be expressed using mean-payoff automaton expressions (together with decidability of the emptiness problem). A detailed comparison with the results of [1] is given in Section 5. Moreover, we provide algorithmic solutions to the quantitative language inclusion and equivalence problems and to distance computation which have no counterpart for non-quantitative languages. Related notions of metrics have been addressed in stochastic games [9] and probabilistic processes [11,19].

2 Mean-Payoff Automaton Expressions

Quantitative languages. A *quantitative language* L over a finite alphabet Σ is a function $L : \Sigma^\omega \to \mathbb{R}$. Given two quantitative languages L_1 and L_2 over Σ, we denote by $\max(L_1, L_2)$ (resp., $\min(L_1, L_2)$, $\mathrm{sum}(L_1, L_2)$ and $-L_1$) the quantitative language that assigns $\max(L_1(w), L_2(w))$ (resp., $\min(L_1(w), L_2(w))$, $L_1(w) + L_2(w)$, and $-L_1(w)$) to each word $w \in \Sigma^\omega$. The quantitative language $-L$ is called the *complement* of L. The $\max$ and $\min$ operators for quantitative languages correspond respectively to the least upper bound and greatest lower bound for the pointwise order $\preceq$ such that $L_1 \preceq L_2$ if $L_1(w) \leq L_2(w)$ for all $w \in \Sigma^\omega$. Thus, they generalize respectively the union and intersection operators for classical boolean languages.

Weighted automata. A $\mathbb{Q}$-*weighted automaton* is a tuple $A = \langle Q, q_I, \Sigma, \delta, \mathsf{wt}\rangle$, where

- Q is a finite set of states, $q_I \in Q$ is the initial state, and Σ is a finite alphabet;
- $\delta \subseteq Q \times \Sigma \times Q$ is a finite set of labelled transitions. We assume that δ is *total*, i.e., for all $q \in Q$ and $\sigma \in \Sigma$, there exists q' such that $(q, \sigma, q') \in \delta$;
- $\mathsf{wt} : \delta \to \mathbb{Q}$ is a *weight* function, where $\mathbb{Q}$ is the set of rational numbers. We assume that rational numbers are encoded as pairs of integers in binary.

We say that A is *deterministic* if for all $q \in Q$ and $\sigma \in \Sigma$, there exists $(q, \sigma, q') \in \delta$ for exactly one $q' \in Q$. We sometimes call automata *nondeterministic* to emphasize that they are not necessarily deterministic.

Words and runs. A *word* $w \in \Sigma^\omega$ is an infinite sequence of letters from Σ. A *lasso-word* w in Σ^ω is an ultimately periodic word of the form $w_1 \cdot w_2^\omega$, where $w_1 \in \Sigma^*$

is a finite prefix, and $w_2 \in \Sigma^+$ is a finite and nonempty word. A *run* of A over an infinite word $w = \sigma_1\sigma_2\dots$ is an infinite sequence $r = q_0\sigma_1 q_1\sigma_2\dots$ of states and letters such that (i) $q_0 = q_I$, and (ii) $(q_i, \sigma_{i+1}, q_{i+1}) \in \delta$ for all $i \geq 0$. We denote by $\mathsf{wt}(r) = v_0 v_1 \dots$ the sequence of weights that occur in r where $v_i = \mathsf{wt}(q_i, \sigma_{i+1}, q_{i+1})$ for all $i \geq 0$.

Quantitative language of mean-payoff automata. The *mean-payoff value* (or limit-average) of a sequence $\bar{v} = v_0 v_1 \dots$ of real numbers is either

$$\mathsf{LimInfAvg}(\bar{v}) = \liminf_{n\to\infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} v_i, \text{ or } \quad \mathsf{LimSupAvg}(\bar{v}) = \limsup_{n\to\infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} v_i.$$

Note that if we delete or insert finitely many values in an infinite sequence of numbers, its limit-averages do not change, and if the sequence is ultimately periodic, then the LimInfAvg and LimSupAvg values coincide (and correspond to the mean of the weights on the periodic part of the sequence). However in general the LimInfAvg and LimSupAvg values do not coincide.

For $\mathsf{Val} \in \{\mathsf{LimInfAvg}, \mathsf{LimSupAvg}\}$, the quantitative language L_A of A is defined by $L_A(w) = \sup\{\mathsf{Val}(\mathsf{wt}(r)) \mid r \text{ is a run of } A \text{ over } w\}$ for all $w \in \Sigma^\omega$. Accordingly, the automaton A and its quantitative language L_A are called LimInfAvg or LimSupAvg. Note that for deterministic automata, we have $L_A(w) = \mathsf{Val}(\mathsf{wt}(r))$ where r is the unique run of A over w.

We omit the weight function wt when it is clear from the context, and we write LimAvg when the value according to LimInfAvg and LimSupAvg coincide (e.g., for runs with a lasso shape).

Decision problems and distance. We consider the following classical decision problems for quantitative languages, assuming an effective presentation of quantitative languages (such as mean-payoff automata, or automaton expressions defined later). Given a quantitative language L and a threshold $\nu \in \mathbb{Q}$, the *quantitative emptiness problem* asks whether there exists a word $w \in \Sigma^\omega$ such that $L(w) \geq \nu$, and the *quantitative universality problem* asks whether $L(w) \geq \nu$ for all words $w \in \Sigma^\omega$.

Given two quantitative languages L_1 and L_2, the *quantitative language-inclusion problem* asks whether $L_1(w) \leq L_2(w)$ for all words $w \in \Sigma^\omega$, and the *quantitative language-equivalence problem* asks whether $L_1(w) = L_2(w)$ for all words $w \in \Sigma^\omega$. Note that universality is a special case of language inclusion where L_1 is constant. Finally, the *distance* between L_1 and L_2 is $D_{\sup}(L_1, L_2) = \sup_{w\in\Sigma^\omega} |L_1(w) - L_2(w)|$. It measures how close is an implementation L_1 as compared to a specification L_2.

It is known that quantitative emptiness is decidable for nondeterministic mean-payoff automata [4], while decidability was open for alternating mean-payoff automata, and for the quantitative language-inclusion problem of nondeterministic mean-payoff automata. From recent undecidability results on games with imperfect information and mean-payoff objective [10] we derive that these problems are undecidable (Theorem 5).

Robust quantitative languages. A class $\mathcal{Q}$ of quantitative languages is *robust* if the class is closed under $\max, \min, \mathrm{sum}$ and complementation operations. The closure properties allow quantitative languages from a robust class to be described compositionally. While nondeterministic LimInfAvg- and LimSupAvg-automata are closed under the $\max$ operation, they are not closed under $\min$ and complement [6]. Alternating $\mathsf{LimInfAvg}$- and $\mathsf{LimSupAvg}$-automata[3] are closed under $\max$ and $\min$, but are not closed under complementation and sum [5]. We define a *robust* class of quantitative languages for mean-payoff automata which is closed under $\max$, $\min$, sum, and complement, and which can express all natural examples of quantitative languages defined using the mean-payoff measure [1,6,7].

Mean-payoff automaton expressions. A *mean-payoff automaton expression* E is obtained by the following grammar rule:

$$E ::= A \mid \max(E, E) \mid \min(E, E) \mid \mathrm{sum}(E, E)$$

where A is a *deterministic* $\mathsf{LimInfAvg}$- or $\mathsf{LimSupAvg}$-automaton. The quantitative language L_E of a mean-payoff automaton expression E is $L_E = L_A$ if $E = A$ is a deterministic automaton, and $L_E = \mathrm{op}(L_{E_1}, L_{E_2})$ if $E = \mathrm{op}(E_1, E_2)$ for $\mathrm{op} \in \{\max, \min, \mathrm{sum}\}$. By definition, the class of mean-payoff automaton expression is closed under $\max$, $\min$ and sum. Closure under complement follows from the fact that the complement of $\max(E_1, E_2)$ is $\min(-E_1, -E_2)$, the complement of $\min(E_1, E_2)$ is $\max(-E_1, -E_2)$, the complement of $\mathrm{sum}(E_1, E_2)$ is $\mathrm{sum}(-E_1, -E_2)$, and the complement of a deterministic $\mathsf{LimInfAvg}$-automaton can be defined by the same automaton with opposite weights and interpreted as a $\mathsf{LimSupAvg}$-automaton, and vice versa, since $-\limsup(v_0, v_1, \dots) = \liminf(-v_0, -v_1, \dots)$. Note that arbitrary linear combinations of deterministic mean-payoff automaton expressions (expressions such as $c_1E_1 + c_2E_2$ where $c_1, c_2 \in \mathbb{Q}$ are rational constants) can be obtained for free since scaling the weights of a mean-payoff automaton by a positive factor $|c|$ results in a quantitative language scaled by the same factor.

3 The Vector Set of Mean-Payoff Automaton Expressions

Given a mean-payoff automaton expression E, let $A_1, \dots, A_n$ be the deterministic weighted automata occurring in E. The *vector set* of E is the set $V_E = \{\langle L_{A_1}(w), \dots, L_{A_n}(w)\rangle \in \mathbb{R}^n \mid w \in \Sigma^\omega\}$ of tuples of values of words according to each automaton A_i. In this section, we characterize the vector set of mean-payoff automaton expressions, and in Section 4 we give an algorithmic procedure to compute this set. This will be useful to establish the decidability of all decision problems, and to compute the distance between mean-payoff automaton expressions. Given a vector $v \in \mathbb{R}^n$, we denote by $\|v\| = \max_i |v_i|$ the ∞-*norm* of v.

The *synchronized product* of $A_1, \dots, A_n$ such that $A_i = \langle Q_i, q_I^i, \Sigma, \delta_i, \mathsf{wt}_i\rangle$ is the $\mathbb{Q}^n$-weighted automaton $A_E = A_1 \times \dots \times A_n = \langle Q_1 \times \dots \times Q_n, (q_I^1, \dots, q_I^n), \Sigma, \delta, \mathsf{wt}\rangle$ such that $t = ((q_1, \dots, q_n), \sigma, (q_1', \dots, q_n')) \in \delta$ if $t_i := (q_i, \sigma, q_i') \in \delta_i$ for all $1 \leq i \leq n$, and $\mathsf{wt}(t) = (\mathsf{wt}_1(t_1), \dots, \mathsf{wt}_n(t_n))$. In the sequel, we assume that all A_i's are deterministic $\mathsf{LimInfAvg}$-automata (hence, A_E is deterministic) and that the underlying graph of the automaton A_E has only one strongly connected component (scc). We show later how to obtain the vector set without these restrictions.

[3] See [5] for the definition of alternating $\mathsf{LimInfAvg}$- and $\mathsf{LimSupAvg}$-automata that generalize nondeterministic automata.

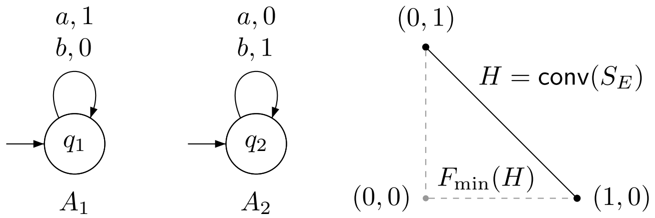


Fig. 1. The vector set of $E = \max(A_1, A_2)$ is $F_{\min}(\mathsf{conv}(S_E)) \supsetneq \mathsf{conv}(S_E)$

For each (simple) cycle ρ in A_E, let the *vector value* of ρ be the mean of the tuples labelling the edges of ρ, denoted $\mathsf{Avg}(\rho)$. To each simple cycle ρ in A_E corresponds a (not necessarily simple) cycle in each A_i, and the vector value $(v_1, \dots, v_n)$ of ρ contains the mean value v_i of ρ in each A_i. We denote by S_E the (finite) set of vector values of simple cycles in A_E. Let $\mathsf{conv}(S_E)$ be the convex hull of S_E.

Lemma 1. *Let E be a mean-payoff automaton expression. The set $\mathsf{conv}(S_E)$ is the closure of the set $\{L_E(w) \mid w$ is a lasso-word$\}$.*

The vector set of E contains more values than the convex hull $\mathsf{conv}(S_E)$, as shown by the following example.

Example 1. Consider the expression $E = \max(A_1, A_2)$ where A_1 and A_2 are deterministic $\mathsf{LimInfAvg}$-automata (see Fig. 1). The product $A_E = A_1 \times A_2$ has two simple cycles with respective vector values $(1, 0)$ (on letter 'a') and $(0, 1)$ (on letter 'b'). The set $H = \mathsf{conv}(S_E)$ is the solid segment on Fig. 1 and contains the vector values of all lasso-words. However, other vector values can be obtained: consider the word $w = a^{n_1} b^{n_2} a^{n_3} b^{n_4} \dots$ where $n_1 = 1$ and $n_{i+1} = (n_1 + \cdots + n_i)^2$ for all $i \geq 1$. It is easy to see that the value of w according to A_1 is 0 because the average number of a's in the prefixes $a^{n_1} b^{n_2} \dots a^{n_i} b^{n_{i+1}}$ for i odd is smaller than $\frac{n_1+\cdots+n_i}{n_1+\cdots+n_i+n_{i+1}} = \frac{1}{1+n_1+\cdots+n_i}$ which tends to 0 when $i \to \infty$. Since A_1 is a $\mathsf{LimInfAvg}$-automaton, the value of w is 0 in A_1, and by a symmetric argument the value of w is also 0 in A_2. Therefore the vector $(0, 0)$ is in the vector set of E. Note that $z = (z_1, z_2) = (0, 0)$ is the pointwise minimum of $x = (x_1, x_2) = (1, 0)$ and $y = (y_1, y_2) = (0, 1)$, i.e. $z = f_{\min}(x, y)$ where $z_1 = \min(x_1, y_1)$ and $z_2 = \min(y_1, y_2)$. In fact, the vector set is the whole triangular region in Fig. 1, i.e. $V_E = \{f_{\min}(x, y) \mid x, y \in \mathsf{conv}(S_E)\}$. □

We generalize $f_{\min}$ to finite sets of points $P \subseteq \mathbb{R}^n$ in n dimensions as follows: $f_{\min}(P) \in \mathbb{R}^n$ is the point $p = (p_1, p_2, \dots, p_n)$ such that p_i is the minimum i^{th} coordinate of the points in P, for $1 \leq i \leq n$. For arbitrary $S \subseteq \mathbb{R}^n$, define $F_{\min}(S) = \{f_{\min}(P) \mid P$ is a finite subset of $S\}$. As illustrated in Example 1, the next lemma shows that the vector set V_E is equal to $F_{\min}(\mathsf{conv}(S_E))$.

Lemma 2. *Let E be a mean-payoff automaton expression built from deterministic* $\mathsf{LimInfAvg}$*-automata, and such that A_E has only one strongly connected component. Then, the vector set of E is $V_E = F_{\min}(\mathsf{conv}(S_E))$.*

For a general mean-payoff automaton expression E (with both deterministic $\mathsf{LimInfAvg}$- and $\mathsf{LimSupAvg}$ automata, and with multi-scc underlying graph), we can use the result of Lemma 2 as follows. We replace each $\mathsf{LimSupAvg}$ automaton A_i occurring in E by the $\mathsf{LimInfAvg}$ automaton A_i' obtained from A_i by replacing every weight wt by $-\mathsf{wt}$. The duality of $\liminf$ and $\limsup$ yields $L_{A_i'} = -L_{A_i}$. In each strongly connected component $\mathcal{C}$ of the underlying graph of A_E, we compute $V_{\mathcal{C}} = F_{\min}(\mathsf{conv}(S_{\mathcal{C}}))$ (where $S_{\mathcal{C}}$ is the set of vector values of the simple cycles in $\mathcal{C}$) and apply the transformation $x_i \rightarrow -x_i$ on every coordinate i where the automaton A_i was originally a $\mathsf{LimSupAvg}$ automaton. The union of the sets $\bigcup_{\mathcal{C}} V_{\mathcal{C}}$ where $\mathcal{C}$ ranges over the strongly connected components of A_E gives the vector set of E.

Theorem 1. *Let E be a mean-payoff automaton expression built from deterministic* $\mathsf{LimInfAvg}$*-automata, and let $\mathcal{Z}$ be the set of strongly connected components in A_E. For a strongly connected component $\mathcal{C}$ let $S_{\mathcal{C}}$ denote the set of vector values of the simple cycles in $\mathcal{C}$. The vector set of E is $V_E = \bigcup_{\mathcal{C} \in \mathcal{Z}} F_{\min}(\mathsf{conv}(S_{\mathcal{C}}))$.*

4 Computation of $F_{\min}(\mathsf{conv}(S))$ for a Finite Set S

It follows from Theorem 1 that the vector set V_E of a mean-payoff automaton expression E can be obtained as a union of sets $F_{\min}(\mathsf{conv}(S))$, where $S \subseteq \mathbb{R}^n$ is a finite set. However, the set $\mathsf{conv}(S)$ being in general infinite, it is not immediate that $F_{\min}(\mathsf{conv}(S))$ is computable. In this section we consider the problem of computing $F_{\min}(\mathsf{conv}(S))$ for a finite set S. In subsection 4.1 we present an explicit construction and in subsection 4.2 we give a geometric construction of the set as a set of linear constraints. We first present some properties of the set $F_{\min}(\mathsf{conv}(S))$.

Lemma 3. *If X is a convex set, then $F_{\min}(X)$ is convex.*

By Lemma 3, the set $F_{\min}(\mathsf{conv}(S))$ is convex, and since $F_{\min}$ is a monotone operator and $S \subseteq \mathsf{conv}(S)$, we have $F_{\min}(S) \subseteq F_{\min}(\mathsf{conv}(S))$ and thus $\mathsf{conv}(F_{\min}(S)) \subseteq F_{\min}(\mathsf{conv}(S))$. The following proposition states that in two dimensions the above sets coincide.

Proposition 1. *Let $S \subseteq \mathbb{R}^2$ be a finite set. Then,* $\mathsf{conv}(F_{\min}(S)) = F_{\min}(\mathsf{conv}(S))$.

We show in the following example that in three dimensions the above proposition does not hold, i.e., we show that $F_{\min}(\mathsf{conv}(S_E)) \neq \mathsf{conv}(F_{\min}(S_E))$ in $\mathbb{R}^3$.

Example 2. We show that in three dimension there is a finite set S such that $F_{\min}(\mathsf{conv}(S)) \not\subseteq \mathsf{conv}(F_{\min}(S))$. Let $S = \{q, r, s\}$ with $q = (0, 1, 0)$, $r = (-1, -1, 1)$, and $s = (1, 1, 1)$. Then $f_{\min}(r, s) = r$, $f_{\min}(q, r, s) = f_{\min}(q, r) = t = (-1, -1, 0)$, and $f_{\min}(q, s) = q$. Therefore $F_{\min}(S) = \{q, r, s, t\}$. Consider $p = (r + s)/2 = (0, 0, 1)$. We have $p \in \mathsf{conv}(S)$ and $f_{\min}(p, q) = (0, 0, 0)$. Hence $(0, 0, 0) \in F_{\min}(\mathsf{conv}(S))$. We now show that $(0, 0, 0)$ does not belong to $\mathsf{conv}(F_{\min}(S))$. Consider $u = \alpha_q \cdot q + \alpha_r \cdot r + \alpha_s \cdot s + \alpha_t \cdot t$ such that u in $\mathsf{conv}(F_{\min}(S))$. Since the third coordinate is non-negative for q, r, s, and t, it follows that if $\alpha_r > 0$ or $\alpha_s > 0$, then the third coordinate of u is positive. If $\alpha_s = 0$ and $\alpha_r = 0$, then we have two cases: (a) if $\alpha_t > 0$, then the first coordinate of u is negative; and (b) if $\alpha_t = 0$, then the second coordinate of u is 1. It follows $(0, 0, 0)$ is not in $\mathsf{conv}(F_{\min}(S))$. □

4.1 Explicit Construction

Example 2 shows that in general $F_{\min}(\mathsf{conv}(S)) \not\subseteq \mathsf{conv}(F_{\min}(S))$. In this section we present an explicit construction that given a finite set S constructs a finite set S' such that (a) $S \subseteq S' \subseteq \mathsf{conv}(S)$ and (b) $F_{\min}(\mathsf{conv}(S)) \subseteq \mathsf{conv}(F_{\min}(S'))$. It would follow that $F_{\min}(\mathsf{conv}(S)) = \mathsf{conv}(F_{\min}(S'))$. Since convex hull of a finite set is computable and $F_{\min}(S')$ is finite, this would give us an algorithm to compute $F_{\min}(\mathsf{conv}(S))$. For simplicity, for the rest of the section we write F for $F_{\min}$ and f for $f_{\min}$ (i.e., we drop the $\min$ from subscript). Recall that $F(S) = \{f(P) \mid P \text{ finite subset of } S\}$ and let $F_i(S) = \{f(P) \mid P \text{ finite subset of } S \text{ and } |P| \leq i\}$. We consider $S \subseteq \mathbb{R}^n$.

Lemma 4. *Let $S \subseteq \mathbb{R}^n$. Then, $F(S) = F_n(S)$ and $F_n(S) \subseteq F_2^{n-1}(S)$.*

Iteration of a construction γ. We will present a construction γ with the following properties: input to the construction is a finite set Y of points, and the output $\gamma(Y)$ satisfies the following properties

1. **(Condition C1).** $\gamma(Y)$ is finite and subset of $\mathsf{conv}(Y)$.
2. **(Condition C2).** $F_2(\mathsf{conv}(Y)) \subseteq \mathsf{conv}(F(\gamma(Y)))$.

Before presenting the construction γ we first show how to iterate the construction to obtain the following result: given a finite set of points X we construct a finite set of points X' such that $F(\mathsf{conv}(X)) = \mathsf{conv}(F(X'))$.

Iterating γ. Consider a finite set of points X, and let $X_0 = X$ and $X_1 = \gamma(X_0)$. Then

$$\mathsf{conv}(X_1) \subseteq \mathsf{conv}(\mathsf{conv}(X_0)) \quad \text{(since by Condition } \mathbf{C1} \text{ we have } X_1 \subseteq \mathsf{conv}(X_0))$$

and hence $\mathsf{conv}(X_1) \subseteq \mathsf{conv}(X_0)$; and

$$F_2(\mathsf{conv}(X_0)) \subseteq \mathsf{conv}(F(X_1)) \qquad \text{(by Condition } \mathbf{C2})$$

For $i \geq 2$, let $X_i = \gamma(X_{i-1})$, and then by iteration we obtain that for X_{n-1} we have

$$(1)\ \mathsf{conv}(X_{n-1}) \subseteq \mathsf{conv}(X_0) \qquad (2)\ F_2^{n-1}(\mathsf{conv}(X_0)) \subseteq \mathsf{conv}(F(X_{n-1}))$$

From (1) and (2) above, along with the aid of Lemma 4 and Lemma 3, we show the following properties:

$$(A)\ F(\mathsf{conv}(X_0)) = F_n(\mathsf{conv}(X_0)) \subseteq F_2^{n-1}(\mathsf{conv}(X_0)) \subseteq \mathsf{conv}(F(X_{n-1}))$$

$$(B)\ \mathsf{conv}(F(X_{n-1})) \subseteq \mathsf{conv}(F(\mathsf{conv}(X_{n-1}))) \subseteq F(\mathsf{conv}(X_0))$$

By (A) and (B) above we have $F(\mathsf{conv}(X_0)) = \mathsf{conv}(F(X_{n-1}))$. Thus given the finite set X, we have the finite set X_{n-1} such that (a) $X \subseteq X_{n-1} \subseteq \mathsf{conv}(X)$ and (b) $F(\mathsf{conv}(X)) = \mathsf{conv}(F(X_{n-1}))$. We now present the construction γ to complete the result.

The construction γ. Given a finite set Y of points $Y' = \gamma(Y)$ is obtained by adding points to Y in the following way:

- For all $1 \leq k \leq n$, we consider all k-dimensional coordinate planes Π supported by a point in Y;
- Intersect each coordinate plane Π with $\mathsf{conv}(Y)$ and the result is a convex polytope Y_Π;
- We add the corners (or extreme points) of each polytope Y_Π to Y.

The proof that the above construction satisfies condition **C1** and **C2** is given in the fuller version [3], and thus we have the following result.

Theorem 2. *Given a finite set $S \subseteq \mathbb{R}^n$ such that $|S| = m$, the following assertion holds: a finite set S' with $|S'| \leq m^{2^n} \cdot 2^{n^2+n}$ can be computed in $m^{O(n \cdot 2^n)} \cdot 2^{O(n^3)}$ time such that (a) $S \subseteq S' \subseteq \mathsf{conv}(S)$ and (b) $F_{\min}(\mathsf{conv}(S)) = \mathsf{conv}(F_{\min}(S'))$.*

4.2 Linear Constraint Construction

In the previous section we presented an explicit construction of a finite set of points whose convex hull gives us $F_{\min}(\mathsf{conv}(S))$. The explicit construction shows interesting properties of the set $F_{\min}(\mathsf{conv}(S))$, however, the construction is inefficient computationally. In this subsection we present an efficient geometric construction for the computation of $F_{\min}(\mathsf{conv}(S))$ for a finite set S. Instead of constructing a finite set $S' \subseteq \mathsf{conv}(S)$ such that $\mathsf{conv}(S') = F_{\min}(\mathsf{conv}(S))$, we represent $F_{\min}(\mathsf{conv}(S))$ as a finite set of linear constraints.

Consider the *positive orthant* anchored at the origin in $\mathbb{R}^n$, that is, the set of points with non-negative coordinates: $\mathbb{R}^n_+ = \{(z_1, z_2, \ldots, z_n) \mid z_i \geq 0, \forall i\}$. Similarly, the *negative orthant* is the set of points with non-positive coordinates, denoted as $\mathbb{R}^n_- = -\mathbb{R}^n_+$. Using vector addition, we write $y + \mathbb{R}^n_+$ for the positive orthant anchored at y. Similarly, we write $x + \mathbb{R}^n_- = x - \mathbb{R}^n_+$ for the negative orthant anchored at x. The positive and negative orthants satisfy the following simple *duality relation*: $x \in y + \mathbb{R}^n_+$ iff $y \in x - \mathbb{R}^n_+$.

Note that $\mathbb{R}^n_+$ is an n-dimensional convex polyhedron. For each $1 \leq j \leq n$, we consider the $(n-1)$-dimensional face $\mathbb{L}_j$ spanned by the coordinate axes except the j^{th} one, that is, $\mathbb{L}_j = \{(z_1, z_2, \ldots, z_n) \in \mathbb{R}^n_+ \mid z_j = 0\}$.

We say that $y + \mathbb{R}^n_+$ is *supported* by X if $(y + \mathbb{L}_j) \cap X \neq \varnothing$ for every $1 \leq j \leq n$. Assuming $y + \mathbb{R}^n_+$ is supported by X, we can construct a set $Y \subseteq X$ by collecting one point per $(n-1)$-dimensional face of the orthant and get $y = f(Y)$. It is also allowed that two faces contribute the same point to Y. Similarly, if $y = f(Y)$ for a subset $Y \subseteq X$, then the positive orthant anchored at y is supported by X. Hence, we get the following lemma.

Lemma 5 (Orthant Lemma). *$y \in F_{\min}(X)$ iff $y + \mathbb{R}^n_+$ is supported by X.*

Construction. We use the Orthant Lemma to construct $F_{\min}(X)$. We begin by describing the set of points y for which the j^{th} face of the positive orthant anchored at y has a non-empty intersection with X. Define $F_j = X - \mathbb{L}_j$, the set of points of the form $x - z$, where $x \in X$ and $z \in \mathbb{L}_j$.

Lemma 6 (Face Lemma). *$(y + \mathbb{L}_j) \cap X \neq \varnothing$ iff $y \in F_j$.*

Proof. Let $x \in X$ be a point in the intersection, that is, $x \in y + \mathbb{L}_j$. Using the duality relation for the $(n-1)$-dimensional orthant, we get $y \in x - \mathbb{L}_j$. By definition, $x - \mathbb{L}_j$ is a subset of $X - \mathbb{L}_j$, and hence $y \in F_j$. □

It is now easy to describe the set defined in our problem statement.

Lemma 7 (Characterization). $F_{\min}(X) = \bigcap_{j=1}^{n} F_j$.

Proof. By the Orthant Lemma, $y \in F_{\min}(X)$ iff $y + \mathbb{R}_+^n$ is supported by X. Equivalently, $(y + \mathbb{L}_j) \cap X \neq \varnothing$ for all $1 \leq j \leq n$. By the Face Lemma, this is equivalent to y belonging to the common intersection of the sets $F_j = X - \mathbb{L}_j$. □

Algorithm for computation of $F_{\min}(\mathsf{conv}(S))$. Following the construction, we get an algorithm that computes $F_{\min}(\mathsf{conv}(S))$ for a finite set S of points in $\mathbb{R}^n$. Let $|S| = m$. We first represent $X = \mathsf{conv}(S)$ as intersection of half-spaces: we require at most m^n half-spaces (linear constraints). It follows that $F_j = X - \mathbb{L}_j$ can be expressed as m^n linear constraints, and hence $F_{\min}(X) = \bigcap_{j=1}^{n} F_j$ can be expressed as $n \cdot m^n$ linear constraints. This gives us the following result.

Theorem 3. *Given a finite set S of m points in $\mathbb{R}^n$, we can construct in $O(n \cdot m^n)$ time $n \cdot m^n$ linear constraints that represent $F_{\min}(\mathsf{conv}(S))$.*

5 Mean-Payoff Automaton Expressions Are Decidable

Several problems on quantitative languages can be solved for the class of mean-payoff automaton expressions using the vector set. The decision problems of quantitative emptiness and universality, and quantitative language inclusion and equivalence are all decidable, as well as questions related to cut-point languages, and computing distance between mean-payoff languages.

Decision problems and distance. From the vector set $V_E = \{\langle L_{A_1}(w), \ldots, L_{A_n}(w)\rangle \in \mathbb{R}^n \mid w \in \Sigma^\omega\}$, we can compute the *value set* $L_E(\Sigma^\omega) = \{L_E(w) \mid w \in \Sigma^\omega\}$ of values of words according to the quantitative language of E as follows. The set $L_E(\Sigma^\omega)$ is obtained by successive application of *min*-, *max*- and *sum-projections* $p_{ij}^{\min}, p_{ij}^{\max}, p_{ij}^{\mathrm{sum}} : \mathbb{R}^k \to \mathbb{R}^{k-1}$ where $i < j \leq k$, defined by

$$p_{ij}^{\min}((x_1, \ldots, x_k)) = (x_1, \ldots, x_{i-1}, \min(x_i, x_j), x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots x_k),$$
$$p_{ij}^{\mathrm{sum}}((x_1, \ldots, x_k)) = (x_1, \ldots, x_{i-1}, \quad x_i + x_j \quad, x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots x_k),$$

and analogously for $p_{ij}^{\max}$. For example, $p_{12}^{\max}(p_{23}^{\min}(V_E))$ gives the set $L_E(\Sigma^\omega)$ of word values of the mean-payoff automaton expression $E = \max(A_1, \min(A_2, A_3))$.

Assuming a representation of the polytopes of V_E as a boolean combination φ_E of linear constraints, the projection $p_{ij}^{\min}(V_E)$ is represented by the formula

$$\psi = (\exists x_j : \varphi_E \wedge x_i \leq x_j) \vee (\exists x_i : \varphi_E \wedge x_j \leq x_i)[x_j \leftarrow x_i]$$

where $[x \leftarrow e]$ is a substitution that replaces every occurrence of x by the expression e. Since linear constraints over the reals admit effective elimination of existential quantification, the formula ψ can be transformed into an equivalent boolean combination of linear constraints without existential quantification. The same applies to max- and sum-projections.

Successive applications of min-, max- and sum-projections (following the structure of the mean-payoff automaton expression E) gives the value set $L_E(\Sigma^\omega) \subseteq \mathbb{R}$ as a boolean combination of linear constraints, hence it is a union of intervals. From this set, it is easy to decide the quantitative emptiness problem and the quantitative universality problem: there exists a word $w \in \Sigma^\omega$ such that $L_E(w) \geq \nu$ if and only if $L_E(\Sigma^\omega) \cap [\nu, +\infty[\neq \varnothing$, and $L_E(w) \geq \nu$ for all words $w \in \Sigma^\omega$ if and only if $L_E(\Sigma^\omega) \cap]-\infty, \nu[= \varnothing$.

In the same way, we can decide the quantitative language inclusion problem "is $L_E(w) \leq L_F(w)$ for all words $w \in \Sigma^\omega$?" by a reduction to the universality problem for the expression $F - E$ and threshold 0 since mean-payoff automaton expressions are closed under sum and complement. The quantitative language equivalence problem is then obviously also decidable.

Finally, the distance between the quantitative languages of E and F can be computed as the largest number (in absolute value) in the value set of $F - E$. As a corollary, this distance is always a rational number.

Comparison with [1]. The work in [1] considers deterministic mean-payoff automata with multiple payoffs. The weight function in such an automaton is of the form $\mathsf{wt} : \delta \rightarrow \mathbb{Q}^d$. The value of a finite sequence $(v_i)_{1 \leq i \leq n}$ (where $v_i \in \mathbb{Q}^d$) is the mean of the tuples v_i, that is a d-dimensional vector $\mathsf{Avg}_n = \frac{1}{n} \cdot \sum_{i=0}^{n-1} v_i$. The "value" associated to an infinite run (and thus also to the corresponding word, since the automaton is deterministic) is the set $Acc \subseteq \mathbb{R}^d$ of accumulation points of the sequence $(\mathsf{Avg}_n)_{n \geq 1}$.

In [1], a query language on the set of accumulation points is used to define *multi-threshold mean-payoff languages*. For $1 \leq i \leq n$, let $p_i : \mathbb{R}^n \rightarrow \mathbb{R}$ be the usual projection along the i^{th} coordinate. A query is a boolean combination of atomic threshold conditions of the form $\min(p_i(Acc)) \sim \nu$ or $\max(p_i(Acc)) \sim \nu$ where $\sim \in \{<, \leq, \geq, >\}$ and $\nu \in \mathbb{Q}$. A word is accepted if the set of accumulation points of its (unique) run satisfies the query. Emptiness is decidable for such multi-threshold mean-payoff languages, by an argument based on the computation of the convex hull of the vector values of the simple cycles in the automaton [1] (see also Lemma 1). We have shown that this convex hull $\mathsf{conv}(S_E)$ is not sufficient to analyze quantitative languages of mean-payoff automaton expressions. It turns out that a richer query language can also be defined using our construction of $F_{\min}(\mathsf{conv}(S_E))$.

In our setting, we can view a d-dimensional mean-payoff automaton A as a product P_A of 2d copies A_t^i of A (where $1 \leq i \leq d$ and $t \in \{\mathsf{LimInfAvg}, \mathsf{LimSupAvg}\}$), where A_t^i assigns to each transition the i^{th} coordinate of the payoff vector in A, and the automaton is interpreted as a t-automaton. Intuitively, the set Acc of accumulation points of a word w satisfies $\min(p_i(Acc)) \sim \nu$ (resp. $\max(p_i(Acc) \sim \nu)$ if and only if the value of w according to the automaton A_t^i for $t = \mathsf{LimInfAvg}$ (resp. $t = \mathsf{LimSupAvg}$) is $\sim \nu$. Therefore, atomic threshold conditions can be encoded as threshold conditions on single variables of the vector set for P_A. Therefore, the vector set computed in Section 4 allows to decide the emptiness problem for multi-threshold mean-payoff languages, by checking emptiness of the intersection of the vector set with the constraint corresponding to the query.

Furthermore, we can solve more expressive queries in our framework, namely where atomic conditions are linear constraints on $\mathsf{LimInfAvg}$- and $\mathsf{LimSupAvg}$-values. For example, the constraint $\mathsf{LimInfAvg}(\mathsf{wt}_1) + \mathsf{LimSupAvg}(\mathsf{wt}_2) \sim \nu$ is simply encoded as $x_k + x_l \sim \nu$ where k, l are the indices corresponding to $A^1_{\mathsf{LimInfAvg}}$ and $A^2_{\mathsf{LimSupAvg}}$ respectively. Note that the trick of extending the dimension of the d-payoff vector with, say $\mathsf{wt}_{d+1} = \mathsf{wt}_1 + \mathsf{wt}_2$, is not equivalent because $\mathsf{Lim}\{\substack{\mathsf{Sup}\\ \mathsf{Inf}}\}\mathsf{Avg}(\mathsf{wt}_1) \pm \mathsf{Lim}\{\substack{\mathsf{Sup}\\ \mathsf{Inf}}\}\mathsf{Avg}(\mathsf{wt}_2)$ is not equal to $\mathsf{Lim}\{\substack{\mathsf{Sup}\\ \mathsf{Inf}}\}\mathsf{Avg}(\mathsf{wt}_1 \pm \mathsf{wt}_2)$ in general (no matter the choice of $\{\substack{\mathsf{Sup}\\ \mathsf{Inf}}\}$ and $\pm$). Hence, in the context of non-quantitative languages our results also provide a richer query language for the deterministic mean-payoff automata with multiple payoffs.

Complexity. All problems studied in this section can be solved easily (in polynomial time) once the value set is constructed, which can be done in quadruple exponential time. The quadruple exponential blow-up is caused by (a) the synchronized product construction for E, (b) the computation of the vector values of all simple cycles in A_E, (c) the construction of the vector set $F_{\min}(\mathsf{conv}(S_E))$, and (d) the successive projections of the vector set to obtain the value set. Therefore, all the above problems can be solved in 4EXPTIME.

Theorem 4. *For the class of mean-payoff automaton expressions, the quantitative emptiness, universality, language inclusion, and equivalence problems, as well as distance computation can be solved in 4EXPTIME.*

Theorem 4 is in sharp contrast with the nondeterministic and alternating mean-payoff automata for which language inclusion is undecidable (see also Table 1). The following theorem presents the undecidability result that is derived from the results of [10].

Theorem 5. *The quantitative universality, language inclusion, and language equivalence problems are undecidable for nondeterministic mean-payoff automata; and the quantitative emptiness, universality, language inclusion, and language equivalence problems are undecidable for alternating mean-payoff automata.*

6 Expressive Power and Cut-Point Languages

We study the expressive power of mean-payoff automaton expressions (i) according to the class of quantitative languages that they define, and (ii) according to their cut-point languages.

Expressive power comparison. We compare the expressive power of mean-payoff automaton expressions with nondeterministic and alternating mean-payoff automata. The results of [6] show that there exist deterministic mean-payoff automata A_1 and A_2 such that $\min(A_1, A_2)$ cannot be expressed by nondeterministic mean-payoff automata. The results of [5] shows that there exists deterministic mean-payoff automata A_1 and A_2 such that $\mathrm{sum}(A_1, A_2)$ cannot be expressed by alternating mean-payoff automata. It follows that there exist languages expressible by mean-payoff automaton expression that cannot be expressed by nondeterministic and alternating mean-payoff automata. In Theorem 6 we show the converse, that is, we show that there exist languages expressible by nondeterministic mean-payoff automata that cannot be expressed by mean-payoff automaton expression. It may be noted that the subclass of mean-payoff automaton expressions that only uses min and max operators (and no sum operator) is a strict subclass of alternating mean-payoff automata, and when only the max operator is used we get a strict subclass of the nondeterministic mean-payoff automata.

Theorem 6. *Mean-payoff automaton expressions are incomparable in expressive power with nondeterministic and alternating mean-payoff automata: (a) there exists a quantitative language that is expressible by mean-payoff automaton expressions, but cannot be expressed by alternating mean-payoff automata; and (b) there exists a quantitative language that is expressible by a nondeterministic mean-payoff automaton, but cannot be expressed by a mean-payoff automaton expression.*

Cut-point languages. Let L be a quantitative language over Σ. Given a threshold $\eta \in \mathbb{R}$, the *cut-point language* defined by (L, η) is the language (i.e., the set of words) $L^{\geq \eta} = \{w \in \Sigma^\omega \mid L(w) \geq \eta\}$. It is known for deterministic mean-payoff automata that the cut-point language may not be ω-regular, while it is ω-regular if the threshold η is *isolated*, i.e. if there exists $\epsilon > 0$ such that $|L(w) - \eta| > \epsilon$ for all words $w \in \Sigma^\omega$ [6].

We present the following results about cut-point languages of mean-payoff automaton expressions. First, we note that it is decidable whether a rational threshold η is an isolated cut-point of a mean-payoff automaton expression, using the value set (it suffices to check that η is not in the value set since this set is closed). Second, isolated cut-point languages of mean-payoff automaton expressions are *robust* as they remain unchanged under sufficiently small perturbations of the transition weights. This result follows from a more general robustness property of weighted automata [6] that extends to mean-payoff automaton expressions: if the weights in the automata occurring in E are changed by at most ϵ, then the value of every word changes by at most $\max(k, 1) \cdot \epsilon$ where k is the number of occurrences of the sum operator in E. Therefore $D_{\sup}(L_E, L_{F^\epsilon}) \to 0$ when $\epsilon \to 0$ where F^ϵ is any mean-payoff automaton expression obtained from E by changing the weights by at most ϵ. As a consequence, isolated cut-point languages of mean-payoff automaton expressions are robust. Third, the isolated cut-point language of mean-payoff automaton expressions is ω-regular. To see this, note that every strongly connected component of the product automaton A_E contributes with a closed convex set to the value set of E. Since the max-, min- and sum-projections are continuous functions, they preserve connectedness of sets and therefore each scc C contributes with an interval $[m_C, M_C]$ to the value set of E. An isolated cut-point η cannot belong to any of these intervals, and therefore we obtain a Büchi-automaton for the cut-point language by declaring to be accepting the states of the product automaton A_E that belong to an scc C such that $m_C > \eta$. Hence, we get the following result.

Theorem 7. *Let L be the quantitative language of a mean-payoff automaton expression. If η is an isolated cut-point of L, then the cut-point language $L^{\geq \eta}$ is ω-regular.*

7 Conclusion and Future Works

We have presented a new class of quantitative languages, the *mean-payoff automaton expressions* which are both robust and decidable (see Table 1), and for which the distance between quantitative languages can be computed. The decidability results come with a high worst-case complexity, and it is a natural question for future works to either improve the algorithmic solution, or present a matching lower bound. Another question of interest is to find a robust and decidable class of quantitative languages based on the discounted sum measure [4].

References

1. Alur, R., Degorre, A., Maler, O., Weiss, G.: On omega-languages defined by mean-payoff conditions. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 333–347. Springer, Heidelberg (2009)
2. Bojanczyk, M.: Beyond omega-regular languages. In: Proc. of STACS. LIPIcs, vol. 3. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2010)
3. Chatterjee, K., Doyen, L., Edelsbrunner, H., Henzinger, T.A., Rannou, P.: Mean-payoff automaton expressions. CoRR, abs/1006.1492 (2010)
4. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 385–400. Springer, Heidelberg (2008)
5. Chatterjee, K., Doyen, L., Henzinger, T.A.: Alternating weighted automata. In: Gębala, M. (ed.) FCT 2009. LNCS, vol. 5699, pp. 3–13. Springer, Heidelberg (2009)
6. Chatterjee, K., Doyen, L., Henzinger, T.A.: Expressiveness and closure properties for quantitative languages. In: Proc. of LICS, pp. 199–208. IEEE, Los Alamitos (2009)
7. Chatterjee, K., Ghosal, A., Henzinger, T.A., Iercan, D., Kirsch, C., Pinello, C., Sangiovanni-Vincentelli, A.: Logical reliability of interacting real-time tasks. In: Proc. of DATE, pp. 909–914. ACM, New York (2008)
8. de Alfaro, L.: How to specify and verify the long-run average behavior of probabilistic systems. In: Proc. of LICS, pp. 454–465. IEEE, Los Alamitos (1998)
9. de Alfaro, L., Majumdar, R., Raman, V., Stoelinga, M.: Game relations and metrics. In: Proc. of LICS, pp. 99–108. IEEE, Los Alamitos (2007)
10. Degorre, A., Doyen, L., Gentilini, R., Raskin, J.-F., Toruńczyk, S.: Energy and mean-payoff games with imperfect information. In: Proc. of CSL. LNCS, Springer, Heidelberg (to appear, 2010)
11. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Metrics for labeled markov systems. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 258–273. Springer, Heidelberg (1999)
12. Droste, M., Gastin, P.: Weighted automata and weighted logics. Theor. Comput. Sci. 380(1-2), 69–86 (2007)
13. Droste, M., Kuich, W., Vogler, H.: Handbook of Weighted Automata. Springer, Heidelberg (2009)
14. Droste, M., Kuske, D.: Skew and infinitary formal power series. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 426–438. Springer, Heidelberg (2003)
15. Ehrenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. Int. Journal of Game Theory 8(2), 109–113 (1979)
16. Kupferman, O., Lustig, Y.: Lattice automata. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 199–213. Springer, Heidelberg (2007)
17. Rabin, M.O.: Probabilistic automata. Information and Control 6(3), 230–245 (1963)
18. Schützenberger, M.P.: On the definition of a family of automata. Information and Control 4(2-3), 245–270 (1961)
19. Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., Carrasco, R.C.: Probabilistic finite-state machines-part I. IEEE Trans. Pattern Anal. Mach. Intell. 27(7), 1013–1025 (2005)
20. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theor. Comput. Sci. 158(1&2), 343–359 (1996)

Obliging Games

Krishnendu Chatterjee[1], Florian Horn[1,2], and Christof Löding[3]

[1] IST Austria (Institute of Science and Technology Austria)
[2] CNRS, LIAFA, Université Paris 7, France
[3] RWTH Aachen, Germany
krish.chat@ist.ac.at, horn@liafa.jussieu.fr,
loeding@informatik.rwth-aachen.de

Abstract. Graph games of infinite length provide a natural model for open reactive systems: one player (Eve) represents the controller and the other player (Adam) represents the environment. The evolution of the system depends on the decisions of both players. The specification for the system is usually given as an ω-regular language L over paths and Eve's goal is to ensure that the play belongs to L irrespective of Adam's behaviour.

The classical notion of winning strategies fails to capture several interesting scenarios. For example, strong fairness (Streett) conditions are specified by a number of request-grant pairs and require every pair that is requested infinitely often to be granted infinitely often: Eve might win just by preventing Adam from making any new request, but a "better" strategy would allow Adam to make as many requests as possible and still ensure fairness.

To address such questions, we introduce the notion of *obliging* games, where Eve has to ensure a strong condition Φ, while always allowing Adam to satisfy a weak condition Ψ. We present a linear time reduction of obliging games with two Muller conditions Φ and Ψ to classical Muller games. We consider obliging Streett games and show they are co-NP complete, and show a natural quantitative optimisation problem for obliging Streett games is in FNP. We also show how obliging games can provide new and interesting semantics for multi-player games.

1 Introduction

Games played on graphs provide a natural theoretical model to study problems in verification, such as synthesis of reactive systems [PR89, RW87], synthesis of systems from specifications [BL69, Chu62], and μ-calculus model-checking [Koz83, Sti01].

The vertices of the graph represent the states of the system, the edges represent transitions, the paths represent behaviours, and the players (Eve and the opponent Adam) represent the controller for the system and its environment, respectively. The goal of the controller is to satisfy a specification (desired set of behaviours) irrespective of the way the environment behaves: the synthesis of such a controller corresponds to the construction of a winning strategy in the graph game.

The class of ω-regular objectives provide a robust specification language to express properties that arise in verification and synthesis of reactive systems [Tho97]. Muller

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 284–296, 2010.

and parity specifications are two canonical ways to specify ω-regular objectives. In the classical study of graph games with ω-regular objectives, the input is a graph game G and an ω-regular objective Φ, and the question is whether there is a winning strategy for a player (Eve) that ensures that Φ is satisfied irrespective of the strategy of the other player (Adam).

A specification Φ often consists of two parts: an assumption Φ_A and a guarantee Φ_G and the specification requires $\Phi_A \rightarrow \Phi_G$. The specification Φ_A typically represents the environment assumption under which the guarantee Φ_G needs to be ensured. A winning strategy for Φ may vacuously satisfy Φ by violating Φ_A, whereas a better strategy would ensure the "strong" specification Φ and allow the "weak" specification Φ_A. For example, consider a Streett (fairness) condition: the fairness condition consists of a set of k request-grant pairs, and requires that every request that appears infinitely often, must be granted infinitely often. A winning strategy may satisfy the fairness conditions by not allowing requests to happen, whereas a better strategy would be as follows: it ensures the strong specification that asks for the satisfaction of the strong fairness specification, and allows for the corresponding weak specification that requires that grants are allowed to happen infinitely often.

To address the question above we consider a very general framework of games with two different levels of specifications: a strong one Φ and a weak one Ψ which are, in general, independent of each other. A "gracious" strategy for Eve must *ensure* the strong specification (in the classical sense), and *allow* the weak one: Adam has the choice to satisfy Ψ. We refer to them as *obliging games*. In the important case of fairness specifications, the weak specification can be self-derived from the fairness specification, and the weak specification requires that the requests are allowed to happen infinitely often. The contribution of our work is as follows:

1. We present a linear time reduction of obliging games (with two Muller conditions) to classical games (with a single Muller condition) such that Eve has a winning strategy in the classical game if, and only if, she has a gracious strategy in the obliging game.
2. We present a detailed analysis of the reduction and memory requirement for obliging games when both specifications are given as parity conditions.
3. In the case of fairness specifications (Streett-generalized Büchi conditions), we show that the problem of the existence of a gracious strategy for Eve is co-NP complete.
   We also study a quantitative optimisation version of this problem and show that it belongs to FNP (functional NP).
4. We also show how our concepts can be extended to multi-player games, leading to new and interesting semantics in the context of verification.

Related work. Our notion of "gracious strategy" can be likened to "permissive strategies", which allow as many behaviours as possible) [BJW02]. In [BJW02] it has been shown that most general strategies can be constructed only for safety conditions, and for parity objectives a strategy that captures behaviour of all memoryless winning strategies was presented. Our work is different as our objectives are more general (Muller), and the goal is to construct a strategy that allows a given objective. Our work is also related to multi-player games on graphs and Nash equilibria [CMJ04, Umm08]. However in Nash equilibria there are no two levels of specifications as considered in obliging games.

2 Definitions

Arenas. A *two-player game arena* A is a triple $(V, V_\circ, E)$ where (V, E) is a finite directed graph without deadlocks (each vertex has at least one outgoing edge) and $V_\circ$ is a subset of V called *Eve's vertices*. The vertices in $v \setminus V_\circ$ are *Adam's vertices* and are usually denoted by $V_\Box$.

Plays and Strategies. A *play* ρ on the arena A is a (possibly infinite) sequence $\rho_1\rho_2\dots$ of vertices such that for all $i < |\rho|$, we have $(\rho_i, \rho_{i+1}) \in E$. The *limit vertices of* ρ, denoted by $\mathrm{Inf}(\rho)$, are the vertices occurring infinitely often in ρ: $\mathrm{Inf}(\rho) = \{q \mid \exists^\infty i, \rho_i = q\}$.

A *strategy* of Eve on the arena A is a function σ from $V^*V_\circ$ to V such that for all $x \in V^*$ and for all $v \in V_\circ$, we have $(v, \sigma(xv)) \in E$. A play ρ is *consistent with* σ (or *a σ-play*) if for all $i < |\rho|$, $\rho_i \in V_\circ \Rightarrow \rho_{i+1} = \sigma(\rho_1 \dots \rho_i)$.

Strategies can also be defined as *strategies with memory*. In this case, σ is a tuple $(M, m_0, \sigma^{\mathrm{u}}, \sigma^{\mathrm{n}})$, where M is the (possibly infinite) set of *memory states*, m_0 is the initial memory content, $\sigma^{\mathrm{u}} : (E \times M) \to M$ is the *memory update* function, and $\sigma^{\mathrm{n}} : (V \times M) \to V$ is the *next-move* function. The memory-update function can naturally be extended from edges to finite sequences of vertices: $\sigma^{\mathrm{u}}_+(v_0v_1 \cdots v_i, m)$ is m if $i = 0$ and $\sigma^{\mathrm{u}}((v_{i-1}, v_i), \sigma^{\mathrm{u}}_+(v_0v_1 \cdots v_{i-1}, m))$ if $i \geq 1$. Using this definition, the next move determined by σ for a play $xv \in V^*V_\circ$ is $\sigma^{\mathrm{n}}(v, m)$, where $m = \sigma^{\mathrm{u}}_+(xv, m_0)$. A strategy is *finite-memory* if M is a finite set, and *memoryless* if M is a singleton. Adam's strategies are defined in a similar way.

Muller conditions. A *Γ-colouring γ of an arena* is a partial function of the edges of A to an arbitrary set of colours Γ. We use partial functions here because this sometimes eases the specification of the winning conditions. However, for formal reasons, we sometimes use a colour "$-$" that corresponds to the undefined value. This colour is not considered when building the limit set of a colour sequence (hence the limit set can be empty).

A *Muller condition Φ on Γ* is a subset of 2^Γ, and a play ρ of A *satisfies Φ* if, and only if, $\mathrm{Inf}(\gamma(\rho)) \in \Phi$. Here, $\gamma(\rho)$ corresponds to the sequence of colours obtained by applying γ to the edges of ρ. This is a finite or infinite sequence over Γ, or an infinite sequence over $\Gamma \cup \{-\}$ using the above convention.

We also consider the usual special cases of Muller conditions (recall that we allow partial colourings):

- the *Büchi condition* is the condition $\{\{\top\}, \{\bot, \top\}\}$ on $\{\bot, \top\}$;
- the *co-Büchi condition* is the condition $\{\emptyset, \{\top\}\}$ on $\{\bot, \top\}$;
- the *k-generalised Büchi condition* is the condition $\{\{1, \dots, k\}\}$ on $\{1, \dots, k\}$;
- the *k-parity condition* is the condition on $\{0, \dots, k-1\}$ containing all and only the subsets whose minimum is even;
- a *k-Streett* condition on Γ is given by a set $\{(R_1, G_1), \dots, (R_k, G_k))\}$ of k request-grant pairs of subsets of Γ. It contains all and only the subsets that for each i either intersect G_i or do not intersect R_i.

In the course of our proofs, it is often useful to consider boolean operations on Muller conditions, in which we interpret negation as complementation and conjunction as Cartesian product: if Φ and Ψ are conditions on Γ_Φ and Γ_Ψ, then $\Phi \wedge \Psi$ is the condition on $\Gamma_\Phi \times \Gamma_\Psi$ which contains all and only the sets whose projection on the first component belongs to Φ, and projection on the second component belongs to Ψ.

Notice that colourings are *partial* functions, so their product may return a colour for only one of the components. We then use the neutral colour "$-$" for the undefined component.

Classical and obliging games. A classical Muller game G on Γ is a triple (A, γ, Φ) where A is an arena, γ is a Γ-colouring of A, and Φ —the *winning* condition— is a Muller condition on Γ. An infinite play ρ of A is winning for Eve if it satisfies Φ. A strategy σ is uniformly winning (resp. winning from a vertex q) for Eve if any σ-play (resp. any σ-play starting in q) is winning for her. A vertex q is winning for Eve if she has a winning strategy from q. The winning region of Eve is the set of vertices winning for her. Adam's winning plays, strategies, vertices, and regions are defined likewise, except that a play is winning for Adam if it does not satisfy Φ.

An obliging game G is a tuple $(A, \gamma_\Phi, \Phi, \gamma_\Psi, \Psi)$, where A is an arena, γ_Φ is a Γ_Φ-colouring, Φ —the *strong* condition— is a Muller condition on Γ_Φ, γ_Ψ is a Γ_Ψ-colouring, and Ψ — the *weak* condition— is a Muller condition on Γ_Ψ. A *uniformly gracious strategy σ for Eve* is such that:

- every infinite σ-play ρ satisfies Φ;
- for any finite σ-play x, there is an infinite σ-play ρ satisfying Ψ such that x is a prefix of ρ .

So, Eve has to *allow* Adam to build a play satisfying Ψ at any position, regardless of what he previously did. However, she does not need to *ensure* Ψ if Adam is not willing to cooperate. Notice that there is no dual notion of spoiling strategy for Adam. In particular, the notion of "determinacy" does not make sense in obliging games, as Adam cannot demonstrate Eve's lack of grace with a single strategy.

We refer to obliging games by the names of the two conditions, with the strong condition first: for example, a parity/Büchi obliging game is an obliging game $G = (A, \gamma_\Phi, \Phi, \gamma_\Psi, \Psi)$, where Φ is a parity and Ψ is a Büchi condition.

Example 1. Consider the parity/parity obliging game in Figure 1. The pairs define the colours of the edge, the first component corresponding to the strong condition (Φ) and the second component to the weak condition (Ψ).

In order to satisfy Φ, a play has to either take the edge (q_4, q_6) infinitely often, or the edge (q_8, q_6) infinitely often and the edge (q_7, q_2) finitely often. To satisfy Ψ, an infinite play has to take the edge (q_7, q_2) infinitely often. In this game Eve has to behave differently depending on whether Adam moves to q_3 or q_4. If the token reaches q_6 coming from q_4, then Eve can safely move to q_7. If the game reaches q_6 coming from q_3, then she can first complete the cycle $q_6 q_5 q_8 q_6$ and then move to q_5 and then to q_0. This strategy can be implemented using memory of size 3 and it is a gracious strategy since each path satisfies Φ and Adam can produce a play satisfying Ψ by always moving to q_4.

It is not difficult to observe that there is no gracious strategy for Eve with memory of size two for this game. □

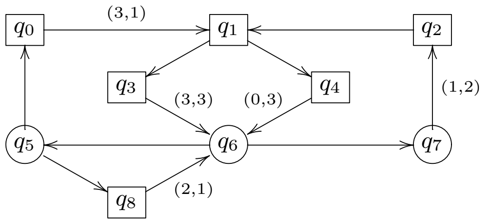


Fig. 1. A parity/parity obliging game

3 Reducing Obliging Games to Classical Games

In this section we provide a general method to reduce obliging games to classical games with a single winning condition. The underlying idea is based on the construction of merciful strategies from [BSL04]: we construct an extended game graph in which Adam decides either to choose his next move himself or to leave this choice to Eve. If he always leaves the choice to Eve from some point onwards, then Eve has to prove that Adam indeed has the possibility to satisfy the weak condition. Consequently, the winning condition for Eve in the new game is the strong condition from the obliging game in conjunction with the weak condition in the case that Adam only finitely often makes his own choice.

Note that in the case that Eve has to satisfy the weak condition, the game remains in a subarena that is completely controlled by Eve. We use this fact by allowing to simplify the weak condition by means of non-deterministic ω-automata. The required technical framework is defined below.

We use ω-automata with an acceptance condition specified on the transitions of the automaton rather than on the states. In our setting, an ω-automaton is of the form $\mathcal{M} = (Q, \Sigma, q_{in}, \Delta, \gamma_\Upsilon, \Upsilon)$, where Q is a finite set of states, Σ is the input alphabet, $q_{in} \in Q$ is the initial state, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $\gamma_\Upsilon : \Delta \to \Gamma_\Upsilon$ is a (partial) colouring function, and Υ is an acceptance condition over Γ_Υ similar to the winning conditions defined for games. We write transitions (q, a, r) with $\gamma_\Upsilon((q, a, r)) = c$ as $q \xrightarrow{a:c} r$.

A run of $\mathcal{M}$ on an infinite word $\alpha \in \Sigma^\omega$ is an infinite sequence $\zeta = q_0 q_1 q_2 \cdots$ of states such that $q_0 = q_{in}$, and $(q_i, \alpha(i), q_{i+1}) \in \Delta$ for each $i \geq 0$. We define the infinite colour sequence induced by α and ζ as the sequence obtained by applying γ_Υ to each transition:

$$\gamma_\Upsilon(\alpha, \zeta) = \gamma_\Upsilon((q_0, \alpha(0), q_1))\gamma_\Upsilon((q_1, \alpha(1), q_2))\gamma_\Upsilon((q_2, \alpha(2), q_3)) \cdots$$

The run ζ on α is accepting if $\gamma_\Upsilon(\alpha, \zeta)$ satisfies the acceptance condition. The language $L(\mathcal{M})$ accepted by $\mathcal{M}$ is the set of all infinite words on which $\mathcal{M}$ has an accepting run.

As usual, we call an automaton deterministic if for each pair of state $q \in Q$ and each $a \in \Sigma$ there is at most one transition $(q, a, r) \in \Delta$.

We are interested in automata accepting languages that correspond to winning conditions. Given a winning condition Φ over Γ_Φ, we define the language $L_\Phi \subseteq (\Gamma_\Phi \cup \{-\})^\omega$

as the set of all infinite sequences that satisfy Φ (recall that "$-$" is a neutral colour representing the undefined value and is not considered for evaluating the winning condition).

Lemma 2. *Let $G = (A, \gamma_\Phi, \Phi, \gamma_\Psi, \Psi)$ be an obliging game with arena $A = (V, E)$, and let $\mathcal{M} = (Q, \Gamma_\Psi, q_{in}, \Delta, \gamma_\Upsilon, \Upsilon)$ be an ω-automaton accepting L_Ψ. There is a game $G' = (A', \gamma_\Lambda, \Lambda)$ and a mapping $\iota : V \to V'$ with the following properties: (1) $\Lambda = \Phi \wedge (\Upsilon \vee B)$ for a Büchi condition B; (2) for each vertex $v \in V$, Eve has a gracious strategy from v in G if, and only if, she has a winning strategy from the vertex $\iota(v)$ in G'; and (3) from a winning strategy for Eve in G' from $\iota(v)$ with memory of size n one can construct a gracious strategy for Eve in G from v with memory of size $2 \cdot |Q| \cdot n$.*

Proof. To simplify the reduction, we assume without loss of generality that the arena is alternating, *i.e.* $E \subseteq (V_\circ \times V_\Box) \cup (V_\Box \times V_\circ)$.

We construct G' in such a way that, at any time in a play, Adam can ask Eve to show a path that satisfies Ψ. This is realised by introducing a second copy of G in which all vertices belong to Eve. In this copy we additionally keep track of the states of the automaton $\mathcal{M}$ recognising Ψ.

If Adam chooses to switch to this copy, Eve makes the choices on behalf of Adam. Consequently, if from some point onward Adam decides to always leave his choices to Eve, the resulting play has to satisfy Φ and Ψ. Otherwise, it is sufficient for Eve to satisfy Φ. The Büchi condition is used to distinguish these two cases. Whether Ψ is satisfied can be decided using the condition Υ on the state sequence of $\mathcal{M}$.

Formally, the game $G' = (A', \gamma_\Lambda, \Lambda)$ and the mapping ι are constructed as follows:

- The winning condition is $\Lambda = \Phi \wedge (\Upsilon \vee B)$ where B is a Büchi condition.
- The arena $A' = (V', V'_\circ, E')$ and the colouring γ_Λ of E' are defined as follows:
  - $V' = (V_\circ \times \{\mathit{play}\}) \cup (V_\Box \times \{\mathit{choose}\} \times Q) \cup (V \times \{\mathit{show}\} \times Q)$;
  - $V'_\circ = (V_\circ \times \{\mathit{play}\}) \cup (V \times \{\mathit{show}\} \times Q)$;
  - Let u and v be vertices in V; q and r be states in Q; and a, b, c be colours in Γ_Φ, Γ_Ψ, Γ_Υ such that $u \xrightarrow{(a,b)} v$ in E and $q \xrightarrow{b:c} r$ in Δ. Then the following edges belong to E':

$$\begin{aligned}
u \in V_\circ :&\ (u, \mathit{play}) \xrightarrow{(a,-,\bot)} (v, \mathit{choose}, q_{in})\\
u \in V_\Box :&\ (u, \mathit{choose}, q) \xrightarrow{(a,-,\top)} (v, \mathit{play})\\
&\ (u, \mathit{choose}, q) \xrightarrow{(-,-,\bot)} (u, \mathit{show}, q)\\
u \in V_\circ :&\ (u, \mathit{show}, q) \xrightarrow{(a,c,\bot)} (v, \mathit{choose}, r)\\
u \in V_\Box :&\ (u, \mathit{show}, q) \xrightarrow{(a,c,\bot)} (v, \mathit{show}, r)
\end{aligned}$$

- The mapping ι maps each $v \in V_\circ$ to (v, play) and each $v \in V_\Box$ to $(v, \mathit{choose}, q_{in})$.

A schematic view of the construction is shown in Figure 2. We refer to the nodes from $V_\circ \times \{\mathit{play}\}$ as the play part of the game, the nodes from $V \times \{\mathit{show}\} \times Q$ as the show part, and the nodes from $V_\Box \times \{\mathit{choose}\} \times Q$ as the choice part.

We start by showing that a gracious strategy σ for Eve in the obliging game G can be used to define a winning strategy for Eve in G': Each play ρ' in G' naturally corresponds to a play ρ in G that is obtained by removing the vertices of the type (v, show, q) for

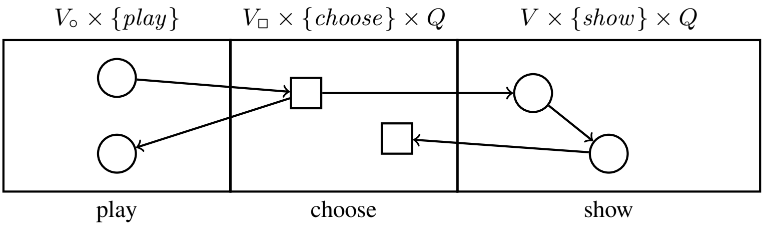

$V_\circ \times \{play\}$
$V_\Box \times \{choose\} \times Q$
$V \times \{show\} \times Q$
play
choose
show

Fig. 2. Schematic view of the reduction from Lemma 2

$v \in V_\Box$ and then projecting away the $\{play, show, choose\}$ and the Q components from the vertices. Let us denote this operation by del, i.e., $\rho = del(\rho')$.

The winning strategy of Eve in G' is defined as follows. For a finite play x' that ends in a node of the form $(u, play)$ with $u \in V_\circ$, Eve looks at the play $del(x')$ in G, checks which move (u, v) she would have made according to σ, and then moves to $(v, choose, q_{in})$ in G'.

If the play x' in G' enters the show part in a node $(u, show, q_{in})$ for the first time after having been in the play part, then Eve considers the play $x = del(x')$ in G. Since σ is a gracious strategy, there is a possible continuation ρ of x such that $x\rho$ is a σ-play satisfying Ψ. In particular, since Ψ is a Muller condition, ρ satisfies Ψ and there is an accepting run ζ of $\mathcal{M}$ on ρ. Eve stores ρ and ζ in her memory for the strategy σ' and now moves from $(u, show, q_{in})$ according to ρ for the first component, and according to ζ for the third component.

If the play x' in G' is in a node $(u, show, q)$ such that Eve has already stored some ρ and ζ in her memory as described above, then she simply moves according to ρ and ζ: she checks at which position in the play she has stored ρ and ζ, which part of ρ and ζ she has already reproduced since then, and makes the corresponding next move to reproduce one more step of ρ and ζ.

If Adam at some point decides to enter the play part, i.e., to move to a vertex from $V_\circ \times \{play\}$, then Eve erases ρ and ζ from her memory.

If π' is an infinite play according to this strategy, then it certainly satisfies Φ because $del(\pi')$ is a σ-play and the Γ_Φ sequence of π' corresponds to the one of $del(\pi')$ except for some insertions of the neutral colour $-$. Furthermore, either Adam infinitely often moves to a vertex from $V_\circ \times \{play\}$, in which case the Büchi condition B is satisfied, or from some point onward Eve simulates ρ and ζ to infinity, yielding a play in G' that satisfies Υ because ζ satisfies Υ. This shows that π' is winning and hence we have defined a winning strategy for Eve, as desired.

For the other direction it suffices to show the third claim of the lemma since the existence of a winning strategy for Eve in G' implies the existence of a finite-memory winning strategy. Let $(M, m_0, \varsigma^{\mathrm{n}}, \varsigma^{\mathrm{u}})$ be a winning strategy for Eve in G'. We define a gracious strategy $(\{\mathsf{p}, \mathsf{s}\} \times Q \times M, (\mathsf{p}, q_{in}, m_0), \sigma^{\mathrm{n}}, \sigma^{\mathrm{u}})$ for Eve in G. This strategy distinguishes two cases to decide whether to use ς^{n} as defined on the $play$ vertices or on the $show$ vertices. These two cases depend on the behaviour of Adam. If Adam makes a move in G from a vertex v that corresponds to the move of ς^{n} from the vertex $(v, show)$ in G', then σ^{u} updates the first component of the memory to s, i.e., σ^{n} starts simulating

ς^{n} as if the play is in the *show* part of G'. If Adam makes a move that is not of this kind, then σ^{u} updates the first component of the memory to p and σ^{n} simulates the behaviour of ς^{n} on the *play* part of G'.

We first give the definition of the next move function σ^{n}, which is quite straightforward:

$$\sigma^{\mathrm{n}}(u, \langle \mathsf{p}, q_{in}, m\rangle) = v \text{ with } \varsigma^{\mathrm{n}}((u, \mathit{play}), m) = (v, \mathit{play}, q_{in}),$$
$$\sigma^{\mathrm{n}}(u, \langle \mathsf{s}, q, m\rangle) = v \text{ if } \varsigma^{\mathrm{n}}((u, \mathit{show}, q), m) = (v, \mathit{play}, q') \text{ for some } q'.$$

The definition of the memory update function σ^{u} is a bit more involved since we have to distinguish the different behaviours of player 1 as explained above. Below, we define the update of the memory for a move from u to v in G for different memory contents. If $u \in V_\circ$, we assume that v is the vertex that is chosen by the next move function σ^{n} because otherwise the move from u to v cannot occur in a play according to the strategy.

(i) If $u \in V_\circ$, then $\sigma^{\mathrm{u}}(u, \langle \mathsf{p}, q_{in}, m\rangle, v) = \langle \mathsf{p}, q_{in}, m'\rangle$ with

$$m' = \varsigma^{\mathrm{u}}((u, \mathit{play}), m, (v, \mathit{choose}, q_{in}))$$

and $\sigma^{\mathrm{u}}(u, \langle \mathsf{s}, q, m\rangle, v) = \langle \mathsf{p}, q', m'\rangle$ with

$$m' = \varsigma^{\mathrm{u}}((u, \mathit{show}, q), m, (v, \mathit{choose}, q'))$$

and $\varsigma^{\mathrm{n}}((u, \mathit{show}, q), m) = (v, \mathit{choose}, q')$ (here we use the assumption that $\sigma^{\mathrm{n}}(u, \langle \mathsf{s}, q, m\rangle) = v$, i.e., v is the target of the next move function).

(ii) If $u \in V_\Box$ and $\varsigma^{\mathrm{n}}((u, \mathit{show}, q), \varsigma^{\mathrm{u}}((u, \mathit{choose}, q), m, (u, \mathit{show}, q))) = (v, \mathit{show}, q')$, then $\sigma^{\mathrm{u}}(u, \langle \mathsf{x}, q, m\rangle, v) = \langle \mathsf{s}, q', m'\rangle$ with

$$m' = \varsigma^{\mathrm{u}}_{+}((u, \mathit{play}, q)(u, \mathit{show}, q)(v, \mathit{show}, q'), m)$$

for all $\mathsf{x} \in \{\mathsf{p}, \mathsf{s}\}$. This is the case where the move from u to v of Adam in G corresponds to the move that Eve would have made in his place in G'. To obtain m' we look at how the memory would have evolved in G' in the move sequence in which Adam gives the choice to Eve.

(iii) If $u \in V_\Box$ and $\varsigma^{\mathrm{n}}((u, \mathit{show}, q), \varsigma^{\mathrm{u}}((u, \mathit{choose}, q), m, (u, \mathit{show}, q))) = (v', \mathit{show}, q')$ for some $v' \neq v$, then $\sigma^{\mathrm{u}}(u, \langle \mathsf{x}, q, m\rangle, v) = \langle \mathsf{p}, q_{in}, m'\rangle$ with

$$m' = \varsigma^{\mathrm{u}}((u, \mathit{choose}, q), m, (v, \mathit{play}))$$

for all $\mathsf{x} \in \{\mathsf{p}, \mathsf{s}\}$. This is the case where Adam makes a choice different from the one that Eve would have made on his behalf in G'.

We now show that this strategy is indeed gracious in G. From the definition of σ^{n} and σ^{u} one can see that for every σ^{n}-play ρ there exists a corresponding ς^{n}-play ρ' that is obtained from *play* by inserting appropriate vertices from $V_\Box \times \{\mathit{show}\} \times Q$ at those positions where σ^{u} updates the first component of the memory to s, i.e., if (ii) in the definition of σ^{u} is applied.

To formalize this let $\rho = v_0 v_1 v_2 \cdots$ be a σ^{n}-play and let

$$\langle \mathsf{x}_0, q_0, m_0\rangle \langle \mathsf{x}_1, q_1, m_1\rangle \langle \mathsf{x}_2, q_2, m_2\rangle \cdots \in (\{\mathsf{p}, \mathsf{s}\} \times Q \times M)^\omega$$

be the corresponding sequence of memory contents according to σ^{u}.

Similar to the operation del from the first implication of the proof we now define an operation ins that transforms ρ into a corresponding play based on the sequence of memory contents. By abuse of notation we also define the operation ins to work on tuples of nodes by inserting the necessary information (we assume for simplicity that the play starts in $V_\circ$):

$$ins(\rho) = (v_0, play)ins(v_0, v_1)ins(v_1, v_2)ins(v_2, v_3)\cdots$$

with

$$ins(v_i, v_{i+1}) = \begin{cases} (v_{i+1}, play) & \text{if } \mathsf{x}_{i+1} = \mathsf{p} \text{ and } v_{i+1} \in V_\circ, \\ (v_{i+1}, choose, q_{i+1}) & \text{if } v_{i+1} \in V_\Box, \\ (v_i, show, q_i)(v_{i+1}, show, q_{i+1}) & \text{if } x_{i+1} = \mathsf{s} \text{ and } v_{i+1} \in V_\circ. \end{cases}$$

Now one can verify that a σ-play ρ in G is transformed by ins into a ς-play ρ' in G'. Therefore, ρ satisfies $\varPhi$ because the colour sequences from $\varGamma_\varPhi$ of ρ and ρ' are the same except for some insertions of the neutral colour $-$. Furthermore, at each position of a play in G, Adam has the possibility to move so that Eve updates her memory content to an element with s in the first component: for a σ-play x in G Adam checks what would have been the move of Eve according to ς in G' for the play $ins(x)$ extended by Adam's move to the show part of the game. If Adam always copies these ς moves to G from some point onwards, then the resulting play ρ satisfies $\varPsi$ because $ins(\rho)$ is a ς-play in G' that does visit $V_\circ \times \{play\}$ only finitely often and hence satisfies $\varUpsilon$. This means that the simulated run of $\mathcal{M}$ on the play is accepting and therefore the corresponding play in G satisfies $\varPsi$. This shows that σ is indeed a gracious strategy. □

Lemma 2 provides a reduction of obliging games to standard games. This notion is formalised as follows. We say that an obliging game G can be reduced to a standard game G' with memory m if:

1. there is a mapping ι from the vertices of G to the vertices of G' such that for each vertex v of G Eve has a gracious strategy from v in G if, and only if, Eve has a winning strategy from $\iota(v)$ in G';
2. given a winning strategy for Eve from $\iota(v)$ in G' with memory of size n, one can compute a gracious strategy for Eve from v in G with memory of size $m \cdot n$.

We also use this notion in connection with classes of games. A class $\mathcal{K}$ of games can be reduced to a class $\mathcal{K}'$ of games with memory m if each game G in $\mathcal{K}$ can be reduced to a game G' in $\mathcal{K}'$ with memory m. The time complexity of such a reduction is the time needed to compute G' from G, to compute the mapping ι, and to compute the strategy in G from the strategy in G'.

We can now instantiate Lemma 2 for several types of obliging games to obtain results on their complexity. The first instantiation is for general Muller conditions using the fact that the winning sequences for a condition $\varPsi$ can be recognised by a one state ω-automaton which itself uses the condition $\varPsi$.

Theorem 3. *There is a linear time reduction with memory* 2 *from* $\varPhi$/$\varPsi$ *obliging games to standard* $(\varPhi \wedge (\varPsi \vee B)))$ *games for a Büchi condition* B.

The point of using a non-deterministic ω-automaton in the formulation of Lemma 2 is illustrated by the following result.

Theorem 4. *There is a polynomial time reduction with memory* $2(\ell+1)k$ *from* $2k$*-parity/*2ℓ*-parity obliging games to standard* $(2k+2)$*-parity games.*

Proof. We apply Lemma 2 with a Büchi automaton accepting L_Ψ for the 2ℓ-parity condition Ψ. Such a Büchi automaton is easily constructed using $(\ell+1)$ states. On the first state the automaton loops and outputs $\perp$ for each input priority. Using the other ℓ states the automaton can guess at any point that $2i$ is the minimal priority which appears infinitely often in the input sequence. It moves to state i and outputs $\top$ whenever priority $2i$ appears on the input. For greater priorities it outputs $\perp$, and for priorities smaller than $2i$ there is no transition. One easily verifies that this automaton accepts L_Ψ.

Lemma 2 yields a reduction with memory $2(\ell+1)$ to a ($2k$-parity $\wedge$ Büchi) game (using the fact that a disjunction of two Büchi conditions is equivalent to a single Büchi condition). Analysing the Zielonka tree [Zie98, DJW97] of a ($2k$-parity $\wedge$ Büchi) condition shows that it has k leafs and the technique from [DJW97] gives a reduction to $2k+2$-parity game with memory k. The composition of these two reductions gives the claimed reduction. One can note that this proof also works if the weak condition is a Rabin condition with ℓ pairs. □

Since parity games are determined with memoryless strategies (see, e.g., [Tho97] or [Zie98]), Theorem 4 directly gives an upper bound on the memory required for a gracious strategy in parity/parity obliging games.

Corollary 5. *If Eve has a gracious strategy in a* $2k$*-parity/*2ℓ*-parity obliging game, then she has a gracious strategy with memory of size at most* $2(\ell+1)k$.

In the case $\ell = 1$, we have rather tight lower bound for the required memory. Indeed, it is possible to construct a $2k$-parity/Büchi obliging game where Eve needs k memory states. The case $k = 6$ is depicted in Figure 3 (in order to improve readability, there are some vertices of Adam from where two edges lead to the same target).

Eve has a gracious strategy with k memory states that works as follows: if Adam just played $2i$, she plays $2i+1$; otherwise, she plays $2(k-1)$. This strategy clearly ensures the parity condition. Furthermore, Adam can get an infinite number of visits to the $\top$ edge, by always answering $2(i-1)$ to $2i+1$.

There is no gracious strategy for Eve with less than k states: as there are k successors of the central vertex, one of them is never visited. Thus, Eve can ultimately not propose the lower ones safely, and either does not guarantee the parity condition or eventually forbids the Büchi condition.

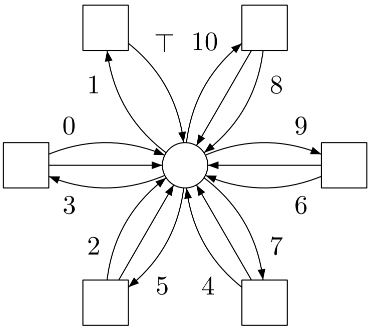


Fig. 3. At least 6 memory states

4 Obliging Streett Games

Streett games are a very natural setting for obligingness questions. Indeed, the Streett condition allows Eve to win by either granting requests or denying Adam the possibility to make them. It is thus natural to consider k-Streett/k-generalised Büchi objectives, where the objectives of the weak condition are exactly the requests of the strong one. We call them simply obliging Streett games. As a generalised Büchi condition can be recognised by a Streett automaton with only one state, we can use Lemma 2 to reduce an obliging k-Streett game with n vertices to a classical $2k$-Streett game with $2n$ vertices. As classical Streett games can also be reduced to obliging Streett games (by always allowing Adam to go to a vertex where all the pairs are forever requested and granted) and classical Streett games problem is co-NP complete [EJ88], it follows that the obliging Streett games problem is co-NP complete:

Theorem 6. *The decision problem of existence of a gracious strategy for Eve in obliging Streett games is co-*NP *complete.*

In the cases where Eve does not have a gracious strategy, we might be interested in knowing how many simultaneous requests she can allow. This can be defined as a threshold problem: "Given $\ell \leq k$, is it possible to allow Adam to visit at least ℓ different requests?"; or as an optimisation problem: "What is the highest ℓ such that Eve can allow Adam to visit at least ℓ different requests?".

Theorem 7. *The threshold problem of obliging Streett games is co-*NP *complete; and the optimisation problem of obliging Streett games is in* FNP.

Proof. As the optimal number of request that Eve can allow is between -1 and k, the second statement follows directly from the first one. Furthermore, it is clear that the threshold problem is co-NP hard since it generalises both classical Streett games (for $\ell = 0$) and obliging Streett games (for $\ell = k$).

In order to show that the threshold problem belongs to co-NP, we use once more Lemma 2: we just need a non-deterministic automaton recognising the words where at least ℓ different colours are visited infinitely often. We describe such an automaton in Figure 4, with the following conventions: the alphabet is $\{1, \ldots, k\}$, and for each i, $R_i = \{i\}$; there is an unmarked loop on each state; unmarked edges are enabled for each letter and are labelled $\perp$. □

5 Multi-player Obliging Games

An interesting feature of obliging games is that they provide new and interesting semantics for multi-player games. In this setting, Eve has more than one opponent and each must be allowed to satisfy his weak condition, regardless of what the others do.

The definitions are similar to the two-player case, *mutatis mutandis*: a *n-player arena* A is a finite directed graph (V, E) without deadlocks whose vertices are partitioned in n subsets, $V_\circ, V_1, \ldots, V_{n-1}$; a *$n$-player obliging game* is a n-player arena and as many colourings and conditions: $\gamma_\circ, \Phi; \gamma_1, \Psi_1; \ldots; \gamma_{n-1}, \Psi_{n-1}$. A gracious strategy σ for Eve in such a game is such that:

- any infinite σ-play ρ satisfies Φ;
- for any $1 \leq i < n$, for any finite σ-play x, there is a strategy τ_i for Player i consistent with x such that any infinite play consistent with both σ and τ_i satisfies Ψ_i.

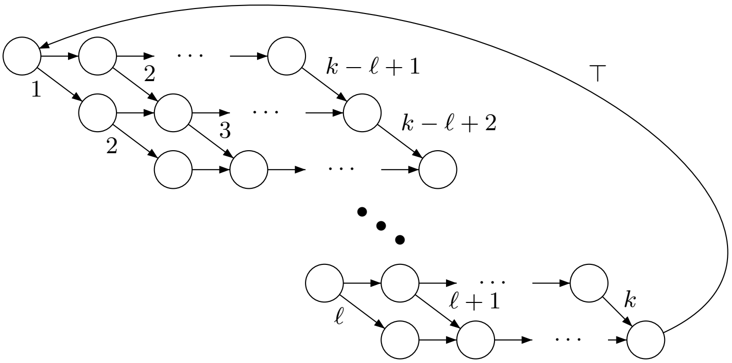


Fig. 4. Büchi automaton recognising repeated ℓ-out-of-k

We can solve n-player obliging games by reduction to classical two-player games, in a way similar to the two-player case. However, we do not use automata to check whether the play satisfies the weak conditions, for two reasons: first, we cannot use non-deterministic automata: even if one opponent yields control of his moves, the others can still interfere so Eve cannot simply "choose" a correct run; second, we would have to remember the current state of each automaton, leading to an exponential blow-up in the size of the arena.

Theorem 8. *Let $G = (A, \gamma_\Phi, \Phi, \gamma_1, \Psi_1, \dots; \gamma_{n-1}, \Psi_{n-1})$ be a n-player obliging game with arena $A = (V, E, V_\circ, V_1, \dots, V_{n-1})$. We can compute, in time linear in the size of G, a game $G' = (A', \gamma_\Upsilon, \Upsilon)$ of size linear in the size of G and a mapping $\iota : V \to V'$ with the following properties:*

1. *$\Upsilon = \Phi \wedge (\Psi_1 \vee B_1) \wedge \dots (\Psi_{n-1} \vee B_{n-1})$, where $B_1, \dots, B_{n-1}$ are Büchi conditions.*
2. *For each vertex v in A, Eve has a gracious strategy from v in G if, and only if, she has a winning strategy from the vertex $\iota(v)$ in G'.*

Proof. The construction of G' is similar to its counterpart in the proof of Lemma 2. Each opponent has the possibility to leave Eve choose his move in his stead. If one of them eventually always does so, the play has to satisfy his weak condition; otherwise, the corresponding Büchi condition allows Eve to ignore it. The proof is even simpler, as there is no need to keep track of a run of an automaton. □

6 Conclusion

In this work we introduced the notion of obliging games and presented a linear time reduction to classical games for all ω-regular objectives specified as Muller objectives. We also presented a complete analysis for the reduction and memory requirement when the specifications are given as parity objectives. We studied the important class of fairness (Streett) conditions, and showed that obligingness Streett games are co-NP complete. We also studied a natural quantitative optimization problem for obliging Streett games and proved inclusion in FNP. We showed extension of the notion of obligingness games to multi-player games and how it leads to new and interesting semantics. In future work we will explore how the solution of obliging games can be used to synthesize more desirable controllers.

References

[BJW02] Bernet, J., Janin, D., Walukiewicz, I.: Permissive strategies: from parity games to safety games. Theoretical Informatics and Applications 36(3), 261–275 (2002)

[BL69] Büchi, J.R., Landweber, L.H.: Solving Sequential Conditions by Finite-State Strategies. Transactions of the AMS 138, 295–311 (1969)

[BSL04] Bontemps, Y., Schobbens, P.-Y., Löding, C.: Synthesis of Open Reactive Systems from Scenario-Based Specifications. Fundamenta Informaticae 62(2), 139–169 (2004)

[Chu62] Church, A.: Logic, arithmetic, and automata. In: Proceedings of the International Congress of Mathematicians, pp. 23–35 (1962)

[CMJ04] Chatterjee, K., Majumdar, R., Jurdziński, M.: On Nash Equilibria in Stochastic Games. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 26–40. Springer, Heidelberg (2004)

[DJW97] Dziembowski, S., Jurdziński, M., Walukiewicz, I.: How Much Memory is Needed to Win Infinite Games? In: Proceedings of LICS, pp. 99–110. IEEE, Los Alamitos (1997)

[EJ88] Emerson, E.A., Jutla, C.S.: The Complexity of Tree Automata and Logics of Programs. In: Proceedings of FOCS, pp. 328–337. IEEE, Los Alamitos (1988)

[Koz83] Kozen, D.: Results on the propositional μ-calculus. TCS 27(3), 333–354 (1983)

[PR89] Pnueli, A., Rosner, R.: On the Synthesis of a Reactive Module. In: Proceedings of POPL, pp. 179–190. ACM, New York (1989)

[RW87] Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete-event processes. SIAM Journal on Control and Optimization 25(1), 206–230 (1987)

[Sti01] Stirling, C.: Modal and Temporal Properties of Processes. Graduate Texts in Computer Science. Springer, Heidelberg (2001)

[Tho97] Thomas, W.: Languages, Automata, and Logic. In: Handbook of Formal Languages. Beyond Words, vol. 3, ch. 7, pp. 389–455. Springer, Heidelberg (1997)

[Umm08] Ummels, M.: The Complexity of Nash Equilibria in Infinite Multiplayer Games. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 20–34. Springer, Heidelberg (2008)

[Zie98] Zielonka, W.: Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. Theoretical Computer Science 200(1-2), 135–183 (1998)

Multipebble Simulations for Alternating Automata
(Extended Abstract)

Lorenzo Clemente and Richard Mayr

LFCS. School of Informatics. University of Edinburgh. UK

Abstract. We study generalized simulation relations for alternating Büchi automata (ABA), as well as alternating finite automata. Having multiple pebbles allows the Duplicator to "hedge her bets" and delay decisions in the simulation game, thus yielding a coarser simulation relation. We define (k_1, k_2)-simulations, with k_1/k_2 pebbles on the left/right, respectively. This generalizes previous work on ordinary simulation (i.e., $(1, 1)$-simulation) for nondeterministic Büchi automata (NBA) in [4] and ABA in [5], and $(1, k)$-simulation for NBA in [3].

We consider direct, delayed and fair simulations. In each case, the (k_1, k_2)-simulations induce a complete lattice of simulations where $(1, 1)$- and (n, n)-simulations are the bottom and top element (if the automaton has n states), respectively, and the order is strict. For any fixed k_1, k_2, the (k_1, k_2)-simulation implies (ω-)language inclusion and can be computed in polynomial time. Furthermore, quotienting an ABA w.r.t. $(1, n)$-delayed simulation preserves its language. Finally, multipebble simulations yield new insights into the Miyano-Hayashi construction [10] on ABA. A technical report with full proofs is available [2].

1 Introduction

We consider simulation relations on (alternating) finite- and infinite word automata: nondeterministic finite automata (NFA), alternating finite automata (AFA), nondeterministic Büchi automata (NBA) and alternating Büchi automata (ABA). Simulation preorder is a notion of semantic comparison of two states, called left state and right state, in automata, where the larger right state can match all moves of the smaller left one in a stepwise way. Simulation preorder implies language inclusion on NFA/AFA/NBA/ABA [4,5], but not vice-versa. While checking language inclusion is PSPACE-complete for all these classes of automata [8,11], the simulation relation can be computed in polynomial time [4,5].

Checking simulation preorder between two states can be presented as a game with two players, Spoiler and Duplicator, where Spoiler tries to prove that the simulation relation does not hold while Duplicator has the opposite objective. In every round of the simulation game, Spoiler chooses a transition from the current left state and Duplicator must choose a transition from the current right state which has the same action label. Duplicator wins iff the game respects the accepting states in the automata, and different requirements for this yield finer or coarser simulation relations. In *direct simulation*, whenever the left state is accepting, the right state must be accepting. In *delayed simulation*, whenever the left state is accepting, the right state must be eventually accepting. In *fair simulation*, if the left state is accepting infinitely often, then the right state must

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 297–312, 2010.

be accepting infinitely often. For finite-word automata, only direct simulation is meaningful, but for Büchi automata delayed and fair simulation yield coarser relations; see [4] for an overview.

These notions have been extended in two directions. Etessami [3] defined a hierarchy of $(1, k)$ multipebble simulations on NBA. Intuitively, the k pebbles on the right side allow Duplicator to “hedge her bets” and thus to delay making decisions. This extra power of Duplicator increases with larger k and yields coarser simulation relations.

A different extension by Wilke and Fritz [5] considered simulations on ABA. In an ABA, a state is either existential or universal. The idea is that Spoiler moves from existential left states and universal right states, and dually for Duplicator.

Our contribution. We consider (k_1, k_2)-simulations on ABA, i.e., with multiple pebbles on both sides: k_1 on the left and k_2 on the right. Intuitively, Duplicator controls pebbles on universal states on the left and existential states on the right (and dually for Spoiler). This generalizes all previous results: the $(1, k)$-simulations on NBA of [3] and the $(1, 1)$-simulations on ABA of [5].

For each acceptance condition (direct, delayed, fair) this yields a lattice-structured hierarchy of (k_1, k_2)-simulations, where $(1, 1)$- and (n, n)-simulations are the bottom and top element if the automaton has n states. Furthermore, the order is strict, i.e., more pebbles make the simulation relation strictly coarser in general. For each fixed $k_1, k_2 \geq 0$, (k_1, k_2)-simulations are computable in polynomial time and they imply language inclusion (over finite or infinite words, depending on the type of simulation).

Quotienting AFA w.r.t. (k_1, k_2)-simulation preserves their language. We also provide a corresponding result for ABA by showing that quotienting ABA w.r.t. $(1, n)$-delayed simulation preserves the ω-language. This is a non-trivial result, since a naïve generalization of the definition of semielective-quotients [5] does not work. We provide the correct notion of semielective-quotients for $(1, n)$-simulations on ABA, and show its correctness. Moreover, unlike for NBA [3], quotienting ABA w.r.t. $(1, k)$ delayed simulation for $1 < k < n$ does *not* preserve their language in general.

Finally, multipebble simulations have close connections to various determinization-like constructions like the subset construction for NFA/AFA and the Miyano-Hayashi construction [10] on ABA. In particular, multipebble simulations yield new insights into the Miyano-Hayashi construction and an alternative correctness proof showing an even stronger property. For full proofs, please see the technical report [2].

2 Preliminaries and Basic Definitions

Automata. An alternating Büchi automaton (ABA) $\mathcal{Q}$ is a tuple $(Q, \Sigma, q_I, \Delta, E, U, F)$, where Q is a finite set of states, Σ is a finite alphabet, q_I is the initial state, $\{E, U\}$ is a partition of Q into *existential* and *universal* states, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation and $F \subseteq Q$ is the set of accepting states. We say that a state q is accepting if $q \in F$. We use n to denote the cardinality of Q. A nondeterministic Büchi automaton (NBA) is an ABA with $U = \emptyset$, i.e., where all choices are existential. We say that $\mathcal{Q}$ is *complete* iff $\forall (q, a) \in Q \times \Sigma .\ \exists (q, a, q') \in \Delta$.

An ABA $\mathcal{Q}$ recognizes a language of infinite words $\mathcal{L}^\omega(\mathcal{Q})$. The acceptance condition is best described in a game-theoretic way [6]. Given an input word $w \in \Sigma^\omega$, the *acceptance game* $\mathbb{G}^\omega(\mathcal{Q}, w)$ is played by two players, Pathfinder and Automaton. Existential states are controlled by Automaton, while Pathfinder controls universal states. Automaton wins the game $\mathbb{G}^\omega(\mathcal{Q}, w)$ iff she has a winning strategy s.t., for any Pathfinder counter-strategy, the resulting computation visits some accepting state in F infinitely often. The language $\mathcal{L}^\omega(\mathcal{Q})$ recognized by $\mathcal{Q}$ is defined as the set of words $w \in \Sigma^\omega$ s.t. Automaton wins $\mathbb{G}^\omega(\mathcal{Q}, w)$. See [5] for a formal definition.

If we view an ABA $\mathcal{Q}$ as an acceptor of *finite* words, then we obtain an alternating finite automaton (AFA). For $w = w_0 \ldots w_m \in \Sigma^*$, the finite acceptance game $\mathbb{G}^{\text{fin}}(\mathcal{Q}, w)$ is defined as above for $\mathbb{G}^\omega(\mathcal{Q}, w)$, except that the game stops when the last symbol w_m of w has been read, and Automaton wins if the last state is in F. $\mathcal{L}^{\text{fin}}(\mathcal{Q})$ is defined in the obvious way. An alternating transition system (ATS) $\mathcal{Q}$ is an AFA where all states are accepting, and $\mathcal{T}r(\mathcal{Q}) := \mathcal{L}^{\text{fin}}(\mathcal{Q})$ is its trace language. When we just say "automaton", it can be an ABA, AFA or ATS, depending on the context.

If Q is a set, with 2^Q we denote the set of subsets of Q, and, for any $k \in \mathbb{N}$, with $2^{Q,k}$ we denote the subset of 2^Q consisting of elements of cardinality at most k. When drawing pictures, we represent existential states by $\textcircled{q}$ and universal states by $\boxed{q}$.

Multipebble simulations. We define multipebble simulations in a game-theoretic way. The game is played by two players, Spoiler and Duplicator, who play in rounds. The objective of Duplicator is to show that simulation holds, while Spoiler has the complementary objective. We use the metaphor of pebbles for describing the game: We call a pebble existential if it is on an existential state, and universal otherwise; *Left* if it is on the l.h.s. of the simulation relation, and *Right* otherwise. Intuitively, Spoiler controls existential *Left* pebbles and universal *Right* pebbles, while Duplicator controls universal *Left* pebbles and existential *Right* pebbles. The presence of > 1 pebbles in each side is due to the further ability of Duplicator to split pebbles to several successors. Moreover, Duplicator always has the possibility of "taking pebbles away". Since not all available pebbles have to be on the automaton, $k + 1$ pebbles are at least as good as k.

Formally, let $\mathcal{Q}$ be an alternating automaton, $\mathbf{q}_0 \in 2^{Q,k_1}$ a k_1-set and $\mathbf{s}_0 \in 2^{Q,k_2}$ a k_2-set. We define the basic (k_1, k_2)-simulation game $\mathbb{G}_{(k_1,k_2)}(\mathbf{q}_0, \mathbf{s}_0)$ as follows. Let Γ^{Sp} and Γ^{Dup} be a set of actions (or transitions) for the two players (to be specified below). In the initial configuration $\langle \mathbf{q}_0, \mathbf{s}_0 \rangle$, *Left* pebbles are on $\mathbf{q}_0$ and *Right* pebbles on $\mathbf{s}_0$. If the current configuration at round i is $\langle \mathbf{q}_i, \mathbf{s}_i \rangle$, then the next configuration $\langle \mathbf{q}_{i+1}, \mathbf{s}_{i+1} \rangle$ is determined as follows:

- Spoiler chooses a transition $(\mathbf{q}_i, \mathbf{s}_i, a_i, \mathbf{q}', \mathbf{s}') \in \Gamma^{\text{Sp}}$.
- Duplicator chooses a transition $(\mathbf{q}_i, \mathbf{s}_i, a_i, \mathbf{q}', \mathbf{s}', \mathbf{q}_{i+1}, \mathbf{s}_{i+1}) \in \Gamma^{\text{Dup}}$.

We now define the two transition relations. Let $\mathbf{q}^E := \mathbf{q} \cap E$ be the set of existential states in $\mathbf{q}$, and define $\mathbf{q}^U, \mathbf{s}^E, \mathbf{s}^U$ similarly. Let $P_1 := 2^{Q,k_1} \times 2^{Q,k_2}$ and $P_0 := \Sigma \times 2^{Q,k_1} \times 2^{Q,k_2}$. $\Gamma^{\text{Sp}} \subseteq P_1 \times P_0$ models Spoiler's moves: $(\mathbf{q}, \mathbf{s}, a, \mathbf{q}', \mathbf{s}') \in \Gamma^{\text{Sp}}$ iff Spoiler chooses a as the next input symbol, and

- $\mathbf{q}'$ is obtained from $\mathbf{q}^E$ by choosing a successor for *each pebble* in $\mathbf{q}^E$. Formally, $\mathbf{q}' = \{ \text{select}(\Delta(q, a)) \mid q \in \mathbf{q}^E \}$, where $\text{select}(\mathbf{r})$ chooses an element in $\mathbf{r}$.
- Similarly, $\mathbf{s}'$ is obtained from $\mathbf{s}^U$ by choosing a successor for each pebble in $\mathbf{s}^U$.

Duplicator's moves are of the form $(\mathbf{q}, \mathbf{s}, a, \mathbf{q}', \mathbf{s}', \mathbf{q}'', \mathbf{s}'') \in \Gamma^{\text{Dup}} \subseteq P_1 \times P_0 \times P_1$:

- $\mathbf{q}''$ is a non-empty k_1-subset of $\mathbf{q}' \cup \Delta(\mathbf{q}^U, a)$, and
- $\mathbf{s}''$ is a non-empty k_2-subset of $\mathbf{s}' \cup \Delta(\mathbf{s}^E, a)$.

Notice that Duplicator is always allowed to "take pebbles away", and to "hedge her bets" by splitting pebbles into different successors. We say that a pebble on state q is *stuck* if q has no a-successor (where a is clear from the context).

We now formally define strategies. A strategy for Spoiler is a function $\delta : P_1^* P_1 \mapsto P_0$ *compatible with* Γ^{Sp}, i.e., for any $(\pi \cdot \langle \mathbf{q}, \mathbf{s} \rangle) \in P_1^* P_1$, $\delta(\pi \cdot \langle \mathbf{q}, \mathbf{s} \rangle) = (a, \mathbf{q}', \mathbf{s}')$ implies $(\mathbf{q}, \mathbf{s}, a, \mathbf{q}', \mathbf{s}') \in \Gamma^{\text{Sp}}$. Similarly, a strategy for Duplicator is a function $\sigma : P_1^* P_1 \mapsto (P_0 \mapsto P_1)$ *compatible with* Γ^{Dup}, i.e., for any $\pi \in P_1^* P_1$ and $(a, \mathbf{q}', \mathbf{s}') \in P_0$, $\sigma(\pi)(a, \mathbf{q}', \mathbf{s}') = \langle \mathbf{q}'', \mathbf{s}'' \rangle$ implies $(\mathbf{q}, \mathbf{s}, a, \mathbf{q}', \mathbf{s}', \mathbf{q}'', \mathbf{s}'') \in \Gamma^{\text{Dup}}$. A play $\pi = \langle \mathbf{q}_0, \mathbf{s}_0 \rangle \langle \mathbf{q}_1, \mathbf{s}_1 \rangle \cdots \in P_1^* \cup P_1^\omega$ is a finite or infinite sequence of configurations in P_1. For a word $w = a_0 a_1 \cdots \in \Sigma^* \cup \Sigma^\omega$ s.t. $|w| = |\pi| - 1$ (with $|\pi| = \omega = \omega - 1$ if $\pi \in \Sigma^\omega$), we say that a play π is *σ-conform to w* iff, for any $i < |\pi|$, there exists some $(\mathbf{q}_i, \mathbf{s}_i, a_i, \mathbf{q}_i', \mathbf{s}_i') \in \Gamma^{\text{Sp}}$ s.t. $\sigma(\langle \mathbf{q}_0, \mathbf{s}_0 \rangle \ldots \langle \mathbf{q}_i, \mathbf{s}_i \rangle)(a_i, \mathbf{q}_i', \mathbf{s}_i') = \langle \mathbf{q}_{i+1}, \mathbf{s}_{i+1} \rangle$. Intuitively, σ-conform plays are those plays which originate when Duplicator's strategy is fixed to σ; δ-conform plays, for δ a Spoiler's strategy, are defined similarly. Below, both strategies are fixed, and the resulting, unique play is conform to both.

The game can halt prematurely, for pebbles may get stuck. In this case, the winning condition is as follows: If there exists a *Left* pebble which cannot be moved, then Duplicator wins. Dually, if no *Right* pebble can be moved, then Spoiler wins.

Remark 1. Our winning condition differs from the one in [5] when pebbles get stuck. There, the losing player is always the one who got stuck. If we let Duplicator win when Spoiler is stuck on a universal *Right* pebble, we would obtain a simulation which *does not imply* language containment. (Notice that "simulation implies containment" is proved in [5] under the assumption that pebbles do not get stuck.) Furthermore, the condition in [5] is unnecessarily strong when Duplicator is stuck on a universal *Left* pebble, where letting Spoiler win is too conservative. Our definition generalizes the correct winning condition to multiple pebbles, for which we prove "simulation implies containment" without further assumptions.

In all other cases, we have that all *Left* pebbles can be moved and at least one *Right* pebble can be moved, and the two players build an infinite sequence of configurations $\pi = \langle \mathbf{q}_0, \mathbf{s}_0 \rangle \langle \mathbf{q}_1, \mathbf{s}_1 \rangle \cdots \in P_1^\omega$. The winning condition is defined in terms of a predicate $C(\pi)$ on π. Different choices of $C(\pi)$ lead to different notions of simulation.

1. *Ordinary (k_1, k_2)-simulation.* The acceptance condition is ignored, and Duplicator wins as long as the game doesn't halt: $C(\pi) :\Longleftrightarrow \text{true}$.
2. *Existential direct (k_1, k_2)-simulation.* Duplicator wins if, whenever *every* $q \in \mathbf{q}_i$ is accepting, then *some* $s \in \mathbf{s}_i$ is accepting:
$$C(\pi) :\Longleftrightarrow (\forall i.\ \mathbf{q}_i \subseteq F \implies \mathbf{s}_i \cap F \neq \emptyset) \ .$$
3. *Universal direct (k_1, k_2)-simulation.* Duplicator wins if, whenever *some* $q \in \mathbf{q}_i$ is accepting, then *every* $s \in \mathbf{s}_i$ is accepting:
$$C(\pi) :\Longleftrightarrow (\forall i.\ \mathbf{q}_i \cap F \neq \emptyset \implies \mathbf{s}_i \subseteq F) \ .$$

As we will see, ordinary simulation is used for ATSs, while existential and universal direct simulation are used for automata over finite and infinite words, respectively.

The winning condition for delayed and fair simulation requires some technical preparation, which consists in the notion of being existentially/universally good since some previous round. Given the current round m, we say that a state $q \in \mathbf{q}_m$ *has seen* a state $\hat{q}$ since some previous round $i \leq m$, written $\mathrm{has\_seen}^i_m(q, \hat{q})$, iff either 1) $q = \hat{q}$, or $i < m$ and there exists $q' \in \mathbf{q}_{m-1}$ s.t. 2.1) $q \in \Delta(q', a_{m-1})$, and 2.2) $\mathrm{has\_seen}^i_{m-1}(q', \hat{q})$. Dually, we write $\mathrm{cant\_avoid}^i_m(q, \hat{q})$ iff either 1) $q = \hat{q}$, or $i < m$ and, for all $q' \in \mathbf{q}_{m-1}$, $q \in \Delta(q', a_{m-1})$ implies $\mathrm{cant\_avoid}^i_{m-1}(q', \hat{q})$. We overload the notation on the set of accepting states, and we write $\mathrm{has\_seen}^i_m(q, F)$ to mean that q has seen some $\hat{q} \in F$; and similarly for $\mathrm{cant\_avoid}^i_m(q, F)$. Finally, we say that $\mathbf{s}_j$ is *existentially good since round* $i \leq j$, written $\mathrm{good}^\exists(\mathbf{s}_j, i)$, if at round j every state in $\mathbf{s}_j$ has seen an accepting state since round i, and j is the least round for which this holds [3]. Similarly, we say that $\mathbf{q}_j$ is *universally good since round* $i \leq j$, written $\mathrm{good}^\forall(\mathbf{s}_j, i)$, if at round j every state in $\mathbf{q}_j$ cannot avoid an accepting state since round i, and j is the least round for which this holds. Formally,

$$\begin{aligned}
\mathrm{good}^\exists(\mathbf{s}_j, i) \iff \quad & (\forall s \in \mathbf{s}_j.\ \mathrm{has\_seen}^i_j(s, F)) \quad \wedge \\
& \forall j'.\ (\forall s' \in \mathbf{s}_{j'}.\ \mathrm{has\_seen}^i_{j'}(s', F)) \implies j' \geq j \\
\mathrm{good}^\forall(\mathbf{s}_j, i) \iff \quad & (\forall s \in \mathbf{s}_j.\ \mathrm{cant\_avoid}^i_j(s, F)) \quad \wedge \\
& \forall j'.\ (\forall s' \in \mathbf{s}_{j'}.\ \mathrm{cant\_avoid}^i_{j'}(s', F)) \implies j' \geq j
\end{aligned}$$

We write $\mathrm{good}^\exists(\mathbf{s}_j)$, omitting the second argument, when we just say that $\mathbf{s}_j$ is good since *some* previous round. For a path $\pi = \mathbf{s}_0\mathbf{s}_1 \ldots$, we write $\mathrm{good}^\exists(\pi, \infty)$, with the second argument instantiated to $i = \infty$, to mean that $\mathrm{good}^\exists(\mathbf{s}_j)$ holds for infinitely many j's; and similarly for $\mathrm{good}^\forall(\mathbf{s}_j)$ and $\mathrm{good}^\forall(\pi, \infty)$.

We are now ready to define delayed and fair simulations.

4. *Delayed* (k_1, k_2)*-simulation.* Duplicator wins if, whenever $\mathbf{q}_i$ is universally good, then there exists $j \geq i$ s.t. $\mathbf{s}_j$ is existentially good since round i:

$$C(\pi) :\iff \forall i.\ \mathrm{good}^\forall(\mathbf{q}_i) \implies \exists j \geq i.\ \mathrm{good}^\exists(\mathbf{s}_j, i)\ .$$

5. *Fair* (k_1, k_2)*-simulation.* Duplicator wins if, whenever there are infinitely many i's s.t. $\mathbf{q}_i$ is universally good, then, for any such i, there exists $j \geq i$ s.t. $\mathbf{s}_j$ is existentially good since round i:

$$C(\pi) :\iff \mathrm{good}^\forall(\pi_0, \infty) \implies (\forall i.\ \mathrm{good}^\forall(\mathbf{q}_i) \implies \exists j \geq i.\ \mathrm{good}^\exists(\mathbf{s}_j, i))\ ,$$

where $\pi_0 = \mathbf{q}_0\mathbf{q}_1 \ldots$ is the projection of π onto its first component.

We will denote the previous acceptance conditions with $x \in \{\mathrm{o}, \exists\mathrm{di}, \forall\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, and the corresponding game is denoted as $\mathbb{G}^x_{(k_1,k_2)}(\mathbf{q_0}, \mathbf{s_0})$.

Remark 2. Notice that the condition for fair simulation is equivalent to the following simpler one: If $\mathbf{q}_i$ is universally good since some previous round infinitely often, then $\mathbf{s}_i$ is existentially good since some previous round infinitely often: $C'(\pi) :\iff \text{good}^{\forall}(\pi_0, \infty) \implies \text{good}^{\exists}(\pi_1, \infty)$, where $\pi_1 = \mathbf{s}_0\mathbf{s}_1 \ldots$ is the projection of π onto its second component.

We are now ready to define the simulation relation $\sqsubseteq^x_{(k_1,k_2)}$, with x as above. We say that a k_2-set $\mathbf{s}$ x-simulates a k_1-set $\mathbf{q}$, written $\mathbf{q} \sqsubseteq^x_{(k_1,k_2)} \mathbf{s}$, if Duplicator has a winning strategy in $\mathbb{G}^x_{(k_1,k_2)}(\mathbf{q}, \mathbf{s})$. We overload the simulation relation $\sqsubseteq^x_{(k_1,k_2)}$ on singletons: $q \sqsubseteq^x_{(k_1,k_2)} s \iff \{q\} \sqsubseteq^x_{(k_1,k_2)} \{s\}$. For two automata $\mathcal{A}$ and $\mathcal{B}$, we write $\mathcal{A} \sqsubseteq^x_{(k_1,k_2)} \mathcal{B}$ for $q_I^{\mathcal{A}} \sqsubseteq^x_{(k_1,k_2)} q_I^{\mathcal{B}}$, where the simulation is actually computed on the disjoint union of $\mathcal{A}$ and $\mathcal{B}$. If $\sqsubseteq^x_{(k_1,k_2)}$ is a simulation, then its transitive closure is defined as $\preceq^x_{(k_1,k_2)}$. Note that, in general, $\sqsubseteq^x_{(k_1,k_2)}$ is not itself a transitive relation.

Multipebble simulations hierarchy. In general, having more pebbles (possibly) gives more power to the Duplicator. This is similar to the $(1,k)$-simulations for NBA studied in [3], but in our context there are two independent directions of "growing power".

Theorem 1. *Let $x \in \{\text{o}, \exists\text{di}, \forall\text{di}, \text{de}, \text{f}\}$ and $k_1' \geq k_1$, $k_2' \geq k_2$.*

1. *Inclusion: $\sqsubseteq^x_{(k_1,k_2)} \subseteq \sqsubseteq^x_{(k_1',k_2')}$. (In particular, $\preceq^x_{(k_1,k_2)} \subseteq \preceq^x_{(k_1',k_2')}$.)*
2. *Strictness: If $k_1' > k_1$ or $k_2' > k_2$, there exists an automaton $\mathcal{Q}'$ s.t. $\sqsubseteq^x_{(k_1,k_2)} \neq \sqsubseteq^x_{(k_1',k_2')}$.*

Proof (Sketch). Point 1) follows directly from the definitions, since Duplicator can always take pebbles away. Point 2) is illustrated in Figure 1, which holds for any kind of simulation $x \in \{\text{o}, \exists\text{di}, \forall\text{di}, \text{de}, \text{f}\}$. □

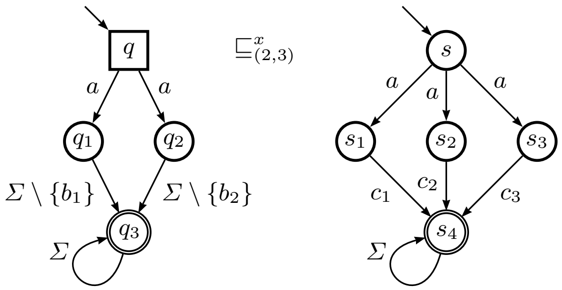


Fig. 1. Example in which $q \sqsubseteq^x_{(2,3)} s$, but $q \not\sqsubseteq^x_{(k_1,k_2)} s$ for any $k_1 \leq 2, k_2 \leq 3$, with $k_1 < 2$ or $k_2 < 3$. The alphabet is $\Sigma' = \{a\} \cup \Sigma$, with $\Sigma = \{b_1, b_2, c_1, c_2, c_3\}$. Note that both automata recognize the same language, both over finite and infinite words: $\mathcal{L}^{\text{fin}}(q) = \mathcal{L}^{\text{fin}}(s) = a(c_1 + c_2 + c_3)\Sigma^*$ and $\mathcal{L}^{\omega}(q) = \mathcal{L}^{\omega}(s) = a(c_1 + c_2 + c_3)\Sigma^{\omega}$.

Theorem 2. *For any $k_1, k_2 \in \mathbb{N}_{>0}$ and any automaton $\mathcal{Q}$,*

1. $\sqsubseteq^{\exists\text{di}}_{(k_1,k_2)} \subseteq \sqsubseteq^{\text{o}}_{(k_1,k_2)}$
2. $\sqsubseteq^{\forall\text{di}}_{(k_1,k_2)} \subseteq \sqsubseteq^{\text{de}}_{(k_1,k_2)} \subseteq \sqsubseteq^{\text{f}}_{(k_1,k_2)} \subseteq \sqsubseteq^{\text{o}}_{(k_1,k_2)}$.

Moreover, for each containment, there exists $\mathcal{Q}$ s.t. the containment is strict.

Proof. The containments follow directly from the definitions. For the strictness, consider again the example in Figure 1, with the modifications below. If no state on the right is accepting, then no simulation holds except ordinary simulation. If q is accepting, then universal direct simulation does not hold, but delayed simulation does. Finally, if the only accepting state is q, then delayed simulation does not hold, but fair simulation does. Is is easy to generalize this example for any $k_1, k_2 \in \mathbb{N}_{>0}$. □

3 Finite Words

Lemma 1. *For any automaton $\mathcal{Q}$ with n states and states $q, s \in Q$:*

1. $q \sqsubseteq^{\exists\mathrm{di}}_{(k_1,k_2)} s$ *implies* $\mathcal{L}^{\mathrm{fin}}(q) \subseteq \mathcal{L}^{\mathrm{fin}}(s)$*, for any* $k_1, k_2 \in \mathbb{N}_{>0}$.
2. $q \sqsubseteq^{\mathrm{o}}_{(k_1,k_2)} s$ *implies* $\mathcal{T}r(q) \subseteq \mathcal{T}r(s)$*, for any* $k_1, k_2 \in \mathbb{N}_{>0}$.
3. $\mathcal{L}^{\mathrm{fin}}(q) \subseteq \mathcal{L}^{\mathrm{fin}}(s)$ *implies* $q \sqsubseteq^{\exists\mathrm{di}}_{(n,n)} s$*, provided that* $\mathcal{Q}$ *is complete.*
4. $\mathcal{T}r(q) \subseteq \mathcal{T}r(s)$ *implies* $q \sqsubseteq^{\mathrm{o}}_{(n,n)} s$*, provided that* $\mathcal{Q}$ *is complete.*

In particular, the last two points above show that existential-direct (resp., ordinary) simulation "reaches" language inclusion (resp., trace inclusion) at (n, n).

Subset constructions. The subset construction is a well-known procedure for determinizing NFAs [8]. It is not difficult to generalize it over *alternating* automata, where it can be used for eliminating existential states, i.e., to perform the *de-existentialization* of the automaton. The idea is the same as in the subset construction, except that, when considering a-successors of a macrostate (for a symbol $a \in \Sigma$), existential and universal states are treated differently. For existential states, we apply the same procedure as in the classic subset construction, by taking always all a-successors. For universal states, each a-successor induces a different transition in the subset automaton. This ensures that macrostates can be interpreted purely disjunctively, and the language of a macrostate equals the union over the language of the states belonging to it. Accordingly, a macrostate is accepting if it contains *some* state which is accepting.

The previous construction can be dualized for de-universalizing finite automata. For an AFA $\mathcal{Q}$, let $\mathcal{S}^{\exists}(\mathcal{Q})$ and $\mathcal{S}^{\forall}(\mathcal{Q})$ be its de-existentialization and de-universalization, respectively. (See Definitions 1 and 2 in Appendix B.1 of the technical report [2].)

The following lemma formalizes the intuition that multipebble simulations for AFA in fact correspond to $(1, 1)$-simulations over the appropriate subset-constructions.

Lemma 2. *Let $\mathcal{Q}_1, \mathcal{Q}_2$ be two AFAs over the same alphabet Σ, with $|Q_1| = n_1$ and $|Q_2| = n_2$. Then, for any $k_1 \leq n_1$ and $k_2 \leq n_2$,*

$$\mathcal{Q}_1 \sqsubseteq^{\exists\mathrm{di}}_{(k_1,n_2)} \mathcal{Q}_2 \iff \mathcal{Q}_1 \sqsubseteq^{\exists\mathrm{di}}_{(k_1,1)} \mathcal{S}^{\exists}(\mathcal{Q}_2) \tag{1}$$

$$\mathcal{Q}_1 \sqsubseteq^{\exists\mathrm{di}}_{(n_1,k_2)} \mathcal{Q}_2 \iff \mathcal{S}^{\forall}(\mathcal{Q}_1) \sqsubseteq^{\exists\mathrm{di}}_{(1,k_2)} \mathcal{Q}_2 \tag{2}$$

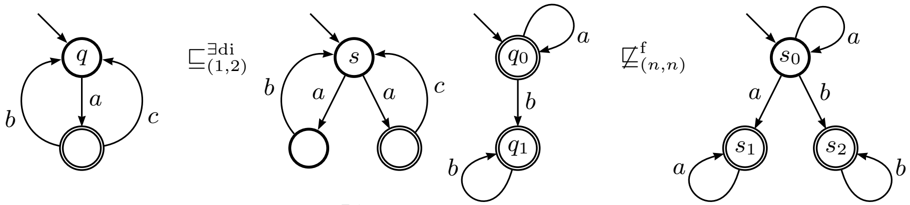
$$\mathcal{Q}_1 \sqsubseteq^{\exists\mathrm{di}}_{(n_1,n_2)} \mathcal{Q}_2 \iff \mathcal{S}^{\forall}(\mathcal{Q}_1) \sqsubseteq^{\exists\mathrm{di}}_{(1,1)} \mathcal{S}^{\exists}(\mathcal{Q}_2)\ . \tag{3}$$

4 Infinite Words

Multipebble existential-direct simulation is not suitable for being used for ω-automata, since it does not even imply ω-language inclusion.

Theorem 3. *For any $k_1, k_2 \in \mathbb{N}_{>0}$, not both equal to 1, there exist an automaton $\mathcal{Q}$ and states $q, s \in \mathcal{Q}$ s.t. $q \sqsubseteq^{\exists di}_{(k_1,k_2)} s$ holds, but $\mathcal{L}^\omega(q) \not\subseteq \mathcal{L}^\omega(s)$.*

Proof. Consider the example in Figure 2(a). Clearly, $q \sqsubseteq^{\exists \mathrm{di}}_{(1,2)} s$ holds, since Duplicator can split pebbles on the successors of s, and one such pebble is accepting, as required by existential-direct simulation. But $\mathcal{L}^\omega(q) \not\subseteq \mathcal{L}^\omega(s)$: In fact, $(ab)^\omega \in \mathcal{L}^\omega(q) = (a(b+c))^\omega$, but $(ab)^\omega \notin \mathcal{L}^\omega(s) = ((ab)^*ac)^\omega$. □



q
a
b
c
$\sqsubseteq^{\exists \mathrm{di}}_{(1,2)}$
s
b
a
a
c
q_0
a
b
q_1
b
$\not\sqsubseteq^{\mathrm{f}}_{(n,n)}$
s_0
a
a
b
s_1
s_2
a
b

(a) An example in which $q \sqsubseteq^{\exists \mathrm{di}}_{(1,2)} s$ holds, but $\mathcal{L}^\omega(q) \not\subseteq \mathcal{L}^\omega(s)$.

(b) An example in which $\mathcal{L}^\omega(q_0) \subseteq \mathcal{L}^\omega(s_0)$ holds, but $q_0 \not\sqsubseteq^{\mathrm{f}}_{(n,n)} s_0$.

Fig. 2. Two examples

This motivates the definition of *universal*-direct simulation, which *does imply ω-language inclusion*, like the coarser delayed and fair simulations.

Theorem 4. *For $x \in \{\forall \mathrm{di}, \mathrm{de}, \mathrm{f}\}$, automaton $\mathcal{Q}$, $k_1, k_2 \in \mathbb{N}_{>0}$ and states $q, s \in \mathcal{Q}$,*

$$q \sqsubseteq^x_{(k_1,k_2)} s \quad \textit{implies} \quad \mathcal{L}^\omega(q) \subseteq \mathcal{L}^\omega(s) \ .$$

Unlike in the finite word case, ω-language inclusion is not "reached" by the simulations $\{\forall \mathrm{di}, \mathrm{de}, \mathrm{f}\}$. See Figure 2(b) and Appendix C in the technical report [2].

Theorem 5. *For any $x \in \{\forall \mathrm{di}, \mathrm{de}, \mathrm{f}\}$, there exist an automaton $\mathcal{Q}$ and states $q_0, s_0 \in \mathcal{Q}$ s.t. $\mathcal{L}^\omega(q_0) \subseteq \mathcal{L}^\omega(s_0)$, but $q_0 \not\sqsubseteq^x_{(n,n)} s_0$.*

The Miyano-Hayashi construction The Miyano-Hayashi (MH) construction [10] is a subset-like construction for ABAs which removes universal non-determinism, i.e., it performs the *de-universalization* of ω-automata. The idea is similar to the analogous construction over finite words, with extra bookkeeping needed for recording visits to accepting states, which may occur not simultaneously for different runs. A set of obligations is maintained, encoding the requirement that, independently of how universal non-determinism is resolved, an accepting state has to be eventually reached. There is a tight relationship between these obligations and fair multipebble simulation. For an ABA $\mathcal{Q}$, let $\mathcal{Q}_{\mathrm{nd}}$ be the de-universalized automaton obtained by applying the MH-construction. (See also Definition 3 in Appendix C.1 of the technical report [2].)

The following lemma says that the MH-construction produces an automaton which is $(n,1)$-fair-simulation equivalent to the original one, and this result is "tight" in the sense that it does not hold for either direct, or delayed simulation.

Lemma 3. *For any ABA $\mathcal{Q}$, let $\mathcal{Q}_{nd}$ be the NBA obtained according to the Miyano-Hayashi de-universalization procedure applied to $\mathcal{Q}$. Then,*

a) $\mathcal{Q} \sqsubseteq^x_{(n,1)} \mathcal{Q}_{nd}$, for $x \in \{\mathrm{f}, \forall\mathrm{di}\}$, and a') $\exists$ automaton $\mathcal{Q}^1$ s.t. $\mathcal{Q}^1 \not\sqsubseteq^{\mathrm{de}}_{(n,1)} \mathcal{Q}^1_{nd}$,
b) $\mathcal{Q}_{nd} \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{Q}$, and b') $\exists$ automaton $\mathcal{Q}^2$ s.t. $\mathcal{Q}^2_{nd} \not\sqsubseteq^x_{(1,1)} \mathcal{Q}^2$, for $x \in \{\mathrm{de}, \forall\mathrm{di}\}$.

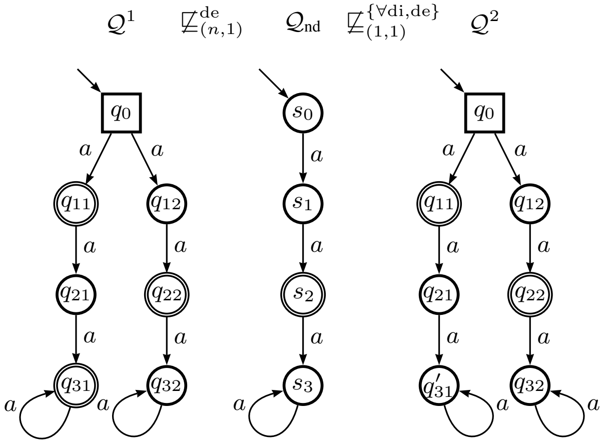


Fig. 3. An example showing automata $\mathcal{Q}^1$ and $\mathcal{Q}^2$ s.t. $\mathcal{Q}^1 \not\sqsubseteq^{\mathrm{de}}_{(n,1)} \mathcal{Q}_{\mathrm{nd}}$ ($n = 2$ suffices), and $\mathcal{Q}_{\mathrm{nd}} \not\sqsubseteq^x_{(1,1)} \mathcal{Q}^2$ for $x \in \{\forall\mathrm{di}, \mathrm{de}\}$. The only difference between $\mathcal{Q}^1$ and $\mathcal{Q}^2$ is the state q_{31} being accepting in the former and q'_{31} being non-accepting in the latter. Notice that $\mathcal{Q}^1_{\mathrm{nd}} = \mathcal{Q}^2_{\mathrm{nd}} = \mathcal{Q}_{\mathrm{nd}}$. The states in $\mathcal{Q}_{\mathrm{nd}}$ are: $s_0 = (\{q_0\}, \{q_0\})$, $s_1 = (\{q_{11}, q_{12}\}, \{q_{12}\})$, $s_2 = (\{q_{21}, q_{22}\}, \emptyset)$, $s_3 = (\{q_{31}, q_{32}\}, \{q_{32}\})$.

Since fair simulation implies language inclusion, $\mathcal{Q}$ and $\mathcal{Q}_{\mathrm{nd}}$ have the same language. This constitutes an alternative proof of correctness for the MH-construction.

The MH-construction "preserves" fair simulation in the following sense.

Lemma 4. *Let $\mathcal{Q}, \mathcal{S}$ be two ABAs. Then, $\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{S} \iff \mathcal{Q}_{nd} \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{S}_{nd}$.*

Remark 3. A weaker version of the "only if" direction of Lemma 4 above, namely $\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{S} \implies \mathcal{Q}_{\mathrm{nd}} \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{S}_{\mathrm{nd}}$ (notice the $(1,1)$ in the premise), had already appeared in [5]. The same statement for both direct and delayed simulation is false, unlike as incorrectly claimed in [5]. In fact, it can be shown (with an example similar to Figure 3) that there exist automata $\mathcal{Q}$ and $\mathcal{S}$ s.t. $\mathcal{Q} \sqsubseteq^x_{(1,1)} \mathcal{S}$, but $\mathcal{Q}_{\mathrm{nd}} \not\sqsubseteq^x_{(1,1)} \mathcal{S}_{\mathrm{nd}}$, with $x \in \{\mathrm{di}, \mathrm{de}\}$. Finally, the "if" direction of Lemma 4 can only be established in the context of multiple pebbles, and it is new.

Transitivity. While most (k_1, k_2)-simulations are not transitive, some limit cases are. By defining a notion of join for $(1, n)$- and $(n, 1)$-strategies (see Appendix C.2 in the technical report [2]), we establish that $(1, n)$ and $(n, 1)$ simulations are transitive.

Theorem 6. *Let $\mathcal{Q}$ be an ABA with n states, and let $x \in \{\forall\mathrm{di}, \mathrm{de}, \mathrm{f}\}$. Then, $\sqsubseteq^x_{(1,n)}$ and $\sqsubseteq^x_{(n,1)}$ are transitive.*

Remark 4 (Difficulties for (n, n) transitivity.). We did consider transitivity for (n, n)-simulations on ABA, but found two major issues there. The first issue concerns directly the definition of the join of two (n, n)-strategies, and this holds for any $x \in \{\forall\mathrm{di}, \mathrm{de}, \mathrm{f}\}$: The so-called "puppeteering technique", currently used for defining the join for $(1, n)$- and $(n, 1)$-strategies, requires to maintain several games, and to pipe the output from one game to the input of one or more other games. This creates a notion of dependency between different games. For $(1, n)$ and $(n, 1)$, there are no cyclic dependencies, and we were able to define the joint strategy. However, for (n, n)-simulations, there are cyclic dependencies, and it is not clear how the joint strategy should be defined.

The second issue arises from the fact that we further require that the join of two winning strategies is itself a winning strategy. Therefore, the joint strategy needs to carry an invariant which implies the x-winning condition, for $x \in \{\forall\mathrm{di}, \mathrm{de}, \mathrm{f}\}$. While such an invariant for $x = \forall\mathrm{di}$ is straightforward, it is not clear what the correct invariant should be for either delayed or fair simulation.

5 Quotienting

In the following we discuss how multipebble simulation preorders can be used for state-space reduction of alternating automata, i.e., we discuss under which notions of quotient the quotient automaton recognizes the same language as the original one.

Let $\mathcal{Q} = (Q, \Sigma, q_I, \Delta, E, U, F)$ be an alternating automaton, over finite or infinite words. Let $\preceq$ be any binary relation on Q, and let $\approx$ be the induced equivalence, defined as $\approx = \preceq^* \cap (\preceq^*)^{-1}$. $[\cdot] : Q \mapsto [Q]$ is the function that maps each element $q \in Q$ to the equivalence class $[q] \in [Q]$ it belongs to, i.e., $[q] := \{q' \mid q \approx q'\}$. We overload $[P]$ on sets $P \subseteq Q$ by taking the set of equivalence classes.

In all the notions of quotients that will be defined, only the transition relation varies. Thus, we gather the common part under a quotient skeleton. We define the *quotient skeleton* $\mathcal{Q}_\approx = ([Q], \Sigma, [q_I], \Delta_\approx, E', U', F')$ as follows: $E' := [E]$, $U' := [Q] \setminus E' = \{ [q] \mid [q] \subseteq U \}$ and $F' = [F]$. We leave $\Delta_\approx$ unspecified at this time, as it will have different concrete instantiations later. Notice that mixed classes, i.e., classes containing both existential and universal states, are declared existential.

The following definitions are borrowed from [5]. We say that $q' \in \Delta(q, a)$ is a *k-x-minimal a-successor of* q iff there there is no strictly $\sqsubseteq^x_{(1,k)}$-smaller a-successor of q, i.e., for any $q'' \in \Delta(q, a)$, $q'' \sqsubseteq^x_{(1,k)} q'$ implies $q' \sqsubseteq^x_{(1,k)} q''$. Similarly, $q' \in \Delta(q, a)$ is a *k-x-maximal a-successor of* q iff for any $q'' \in \Delta(q, a)$, $q' \sqsubseteq^x_{(1,k)} q''$ implies $q'' \sqsubseteq^x_{(1,k)} q'$. Let $\min^{k,x}_a(q)$/$\max^{k,x}_a(q)$ be the set of minimal/maximal successors.

5.1 Finite Words

Let $\preceq$ be *any* preorder which implies language inclusion over finite words, i.e., $q \preceq s \implies \mathcal{L}^{\mathrm{fin}}(q) \subseteq \mathcal{L}^{\mathrm{fin}}(s)$. In particular, one can take $\preceq = (\sqsubseteq^{\exists\mathrm{di}}_{(k_1,k_2)})^*$, or even $\preceq$ equal to language inclusion itself. As before, let $\approx$ be the equivalence induced by $\preceq$. It is well known that automata over finite words can be quotiented w.r.t. any preorder which implies language equivalence. Here, we show that not all transitions are needed, and that is is sufficient to consider $\preceq$-maximal successors of existential states and $\preceq$-minimal successors of universal states. We define the *minimax* [5] quotient automaton $\mathcal{Q}^{\mathrm{m}}_{\approx}$ by instantiating the quotient skeleton (see Section 5) with transition relation $\Delta_{\approx} := \Delta^{\mathrm{m}}_{\approx}$, where $([q], a, [q']) \in \Delta^{\mathrm{m}}_{\approx}$ iff either

- $[q] \in E'$ and $\exists\, \widehat{q} \in [q] \cap E, \widehat{q}' \in [q']$ s.t. $(\widehat{q}, a, \widehat{q}') \in \Delta \wedge \widehat{q}' \in \max^{\preceq}_{a}(\widehat{q})$, or
- $[q] \in U'$ and $\exists\, \widehat{q} \in [q], \widehat{q}' \in [q']$ s.t. $(\widehat{q}, a, \widehat{q}') \in \Delta$ and $\widehat{q}' \in \min^{\preceq}_{a}(\widehat{q})$.

Notice that transitions from universal states in mixed classes are ignored altogether.

Lemma 5. *Let $\mathcal{Q}$ be any alternating finite automaton, and let $\preceq$ be any preorder which implies finite-language inclusion. Then, for any $q \in Q$, $\mathcal{L}^{\mathrm{fin}}(q) = \mathcal{L}^{\mathrm{fin}}([q]_m)$.*

5.2 Infinite Words

Unlike for finite words, it is well known that quotienting ω-automata w.r.t. ω-language-equivalence does not preserve the ω-language. It has even been shown that quotienting w.r.t. $(1, 1)$-fair (bi)simulation does not preserve the ω-language either [7,4]. Therefore, one has to look for finer simulations, like delayed or direct simulation. Notice that multipebble *existential*-direct simulation cannot be used for quotienting, since it does not even imply ω-language inclusion—see Theorem 3.

Theorem 7. *For any $k_1, k_2 \in \mathbb{N}^{>0}$ and $x \in \{\exists\mathrm{di},\ f\}$ there exists an ABA $\mathcal{Q}$ s.t. $\mathcal{L}^{\omega}(\mathcal{Q}) \neq \mathcal{L}^{\omega}(\mathcal{Q}_{\approx})$, with $\approx := \approx^{x}_{(k_1,k_2)}$. For $x = \exists\mathrm{di}$, k_1 and k_2 must not be both equal to 1. (Note that $\approx^{\exists\mathrm{di}}_{(1,1)}$-quotienting* does preserve *the ω-language.)*

Thus, in the following we concentrate on *universal*-direct and delayed simulation.

Minimax quotients for universal-direct simulation. In [5] it has been shown that minimax quotients preserve the ω-language (for direct simulation), and that one can consider just maximal/minimal successors of existential/universal states, respectively. Here, we improve this notion, by showing that, when considering multiple-pebbles, it is not needed to consider *every* maximal successor of existential states, but it is safe to discard those maximal successors which are $(1, k)$-simulated by a *k-set* of other maximal successors. This suggests the following definition: For $\widehat{q} \in E$, $a \in \Sigma$ and $k > 0$, we say that $\widehat{\mathbf{q}}'$ is a set of *k-maximal representatives for a-successors of $\widehat{q}$* iff

$$\widehat{\mathbf{q}}' \subseteq \max^{k,\forall\mathrm{di}}_{a}(\widehat{q}) \wedge \Big(\forall q'' \in \big(\max^{k,\forall\mathrm{di}}_{a}(\widehat{q}) \setminus \widehat{\mathbf{q}}'\big) \,.\, \exists \widehat{\mathbf{q}}'' \in 2^{\widehat{\mathbf{q}}',k} \,.\, q'' \sqsubseteq^{\forall\mathrm{di}}_{(1,k)} \widehat{\mathbf{q}}''\Big) \qquad (4)$$

Notice that the above definition is non-deterministic, in the sense that there might be different sets of maximal representatives: In this case, one can just take any $\subseteq$-minimal set satisfying Equation 4. In the following, we assume that a set of maximal representatives $\widehat{\mathbf{q}}'$ has been selected for any $\widehat{q} \in E$ and $a \in \Sigma$.

We define the *minimax+* quotient automaton $\mathcal{Q}^{\mathrm{m+}}_{\approx}$ by instantiating the quotient skeleton (see Section 5) with transition relation $\Delta_{\approx} := \Delta^{\mathrm{m+}}_{\approx}$, which differs from $\Delta^{\mathrm{m}}_{\approx}$ just for existential and mixed classes: $([q], a, [q']) \in \Delta^{\mathrm{m+}}_{\approx}$ with $[q] \in E'$ iff

- there exist $\widehat{q} \in [q] \cap E$ and $\widehat{q}' \in [q']$ s.t. $(\widehat{q}, a, \widehat{q}') \in \Delta$ and $\widehat{q}' \in \widehat{\mathbf{q}}'$, where $\widehat{\mathbf{q}}'$ is a fixed set of k-maximal representatives for a-successors of $\widehat{q}$, as defined above.

Our definition of minimax+ quotient differs from the one in [5] also w.r.t. the treatment of mixed classes, as discussed in the following remarks.

Remark 5. While in [5] universal states in mixed classes do induce transitions (to minimal elements), in our definition we ignore these transitions altogether. In the setting of $(1, 1)$-simulations these two definitions coincide, as they are shown in [5] to yield exactly the same transitions, but this needs not be the case in our setting: In the context of multiple-pebbles, one minimal transition from a universal state q^U might be subsumed by no single transition from some existential state q^E in the same class, but it is always the case that q^E has a set of transitions which *together* subsume the one from q^U (cf. Lemma 15 in Appendix D.3 [2]). In this case, we show that one can in fact always discard the transitions from q^U. Thus, in the context of multiple-pebbles, minimax+ quotients result in less transitions than just minimax quotients from [5].

Remark 6. While minimax mixed classes are deterministic when considering $(1, 1)$-simulations [5], this is not necessarily true when multiple pebbles are used.

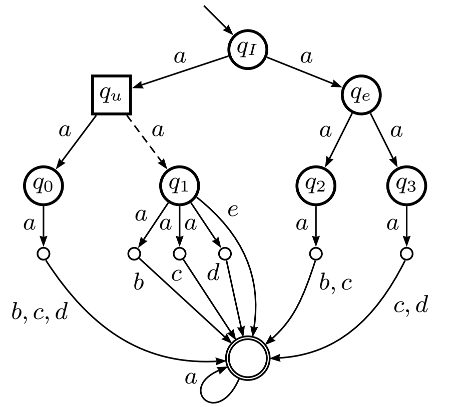
Theorem 8. *$q \approx^{\forall \mathrm{di}}_{(1,n)} [q]_{\mathrm{m+}}$, where the quotient is taken w.r.t. the transitive closure of $\sqsubseteq^{\forall \mathrm{di}}_{(1,k)}$, for any k such that $1 \leq k \leq n$. In particular, $\mathcal{L}^\omega(q) = \mathcal{L}^\omega([q]_{\mathrm{m+}})$.*

Semielective quotients for delayed simulation. It has been shown in [5] that minimax quotients w.r.t $(1, 1)$-delayed simulation on ABA do not preserve the ω-language. The reason is that taking just maximal successors of existential states is incorrect for delayed simulation, since a visit to an accepting state might only occur by performing a non-maximal transition. (This is not the case with direct simulation, where if a simulation-smaller state is accepting, then every bigger state is accepting too.) This motivates the definition of *semielective quotients* [5], which are like minimax quotients, with the only difference that *every* transition induced by existential states is considered, not just maximal ones. Except for that, all previous remarks still apply. In particular, in mixed classes in semielective quotients it is necessary to ignore non-minimal transitions from universal states—the quotient automaton would recognize a bigger language otherwise.

While for the $(1, 1)$-simulations on ABA in [5] it is actually possible to ignore transitions from universal states in mixed classes altogether (see Remark 5), in the context of multiple-pebbles this is actually incorrect (see Figure 5 in Appendix D.3 of the technical report [2]). The reason is similar as why non-maximal transitions from existential states cannot be discarded: This might prevent accepting states from being visited. We define the *semielelective+* quotient automaton $\mathcal{Q}^{\mathrm{se+}}_{\approx}$ by instantiating the quotient skeleton (see Section 5) with $\Delta_{\approx} := \Delta^{\mathrm{se+}}_{\approx}$, where

$$([q], a, [q']) \in \Delta^{\mathrm{se+}}_{\approx} \iff (q, a, q') \in \Delta \text{ and either } q \in E, \text{ or } q \in U \text{ and } q' \in \min^{n,\mathrm{de}}_a(q)$$

Theorem 9. *$q \approx^{\mathrm{de}}_{(1,n)} [q]_{\mathrm{se+}}$, where the quotient is taken w.r.t. $\sqsubseteq^{\mathrm{de}}_{(1,n)}$. In particular, $\mathcal{L}^\omega(q) = \mathcal{L}^\omega([q]_{\mathrm{se+}})$.*



$q_u \approx^{\text{de}}_{(1,2)} q_e$
$q_0 \sqsubseteq^{\text{de}}_{(1,3)} q_1$
$q_0 \not\sqsubseteq^{\text{de}}_{(1,2)} q_1$
$q_1 \not\sqsubseteq^{\text{de}}_{(1,3)} q_0$

Fig. 4. $(1, k)$-semielective+ quotients on ABA do not preserve the ω-language for $1 < k < n$ in general. Let $k = 2$. The only two $(1, k)$/$(1, n)$-equivalent states are q_u and q_e, and in the quotient they form a mixed class. q_1 *is not* a $(1, n)$-minimal a-successor of q_u, but it is a $(1, k)$-minimal successor for $k = 2$. Thus, the only difference between the $(1, n)$- and $(1, k)$-semielective+ quotients is that the dashed transition is (correctly) not included in the former, but (incorrectly) included in the latter. Thus the $(1, k)$-semielective+ quotient automaton would incorrectly accept the word $w = aaea^\omega \notin \mathcal{L}^\omega(q_I) = aaa\{b + c + d\}a^\omega$.

Remark 7. It is surprising that, unlike for NBA [3], quotienting ABA w.r.t. $(1, k)$-de simulations, for $1 < k < n$, does not preserve the language of the automaton in general. The problem is again in the mixed classes, where minimal transitions from universal states can be selected only by looking at the full $(1, n)$-simulation. See the counterexample in Figure 4, where the dashed transition is present in the $(1, k)$-quotient, despite being non-$(1, n)$-minimal.

Remark 8. Semielective multipebble quotients can achieve arbitrarily high compression ratios relative to semielective 1-pebble quotients, (multipebble-)direct minimax quotients and mediated preorder quotients [1] (see Figure 6 in Appendix D.3 [2]).

6 Solving Multipebble Simulation Games

In this section we show how to solve the multipebble simulation games previously defined. We encode each simulation game into a 2-player game-graph with an ω-regular winning condition. In the game-graph, Eve will take the rôle of Duplicator, and Adam the one of Spoiler. A game-graph is a tuple $\mathcal{G} = \langle V_\text{E}, V_\text{A}, \rightarrow\rangle$, where nodes in V_E belong to Eve (mimicking Duplicator), and nodes in V_A belong to Adam (mimicking Spoiler). Transitions are represented by elements in $\rightarrow \subseteq (V_\text{E} \times V_\text{A} \cup V_\text{A} \times V_\text{E})$, where we write $p \rightarrow q$ for $(p, q) \in \rightarrow$. Notice that the two players strictly alternate while playing, i.e., the game graph is *bipartite*. We write V for $V_\text{E} \cup V_\text{A}$. We introduce the following monotone operator on 2^{V_A}: For any $\mathbf{x} \subseteq V_\text{A}$, $\text{cpre}(\mathbf{x}) := \{v_0 \in V_\text{A} \mid \forall v_1 \in V_\text{E}.\ (v_0 \rightarrow v_1 \implies \exists v_2 \in \mathbf{x}.\ v_1 \rightarrow v_2)\}$, i.e., $\text{cpre}(\mathbf{x})$ is the set of nodes where Eve can force the game into $\mathbf{x}$.

We define various game-graphs for solving simulations. We express the winning region of Eve as a μ-calculus fixpoint expression over V_{A} [9], which can then be evaluated using standard fixpoint algorithms. We derive the desired complexity upper bounds using the following fact:

Lemma 6. *Let e be a fixpoint expression over a graph V, with $|V| \in n^{O(k)}$. Then, for any fixed $k \in \mathbb{N}$, evaluating e can be done in time polynomial in n.*

For solving direct and fair simulation, we refer the reader to Appendix E [2]. Here, we consider just delayed simulation, which is the most difficult (and interesting).

The natural starting point for defining $\mathcal{G}^{\mathrm{de}}$ is the definition in [3] of the game-graph for computing $(1, k)$-simulations for NBAs. Unfortunately, the game-graph in [3] is actually incorrect: According to the definition of delayed simulation (cf. Section 2), every new obligation encountered when the left side is accepting at some round should be *independently* satisfied by the right side, which has to be good since *that* round. Now, the algorithm in [3] just tries to satisfy the most recent obligation, which overrides all the previous ones. This is an issue: If the left side is continuously accepting, for example, then the right side might simply have not enough time to satisfy any obligation at all. Therefore, [3] actually computes an *under-approximation* to delayed simulation.

We overcome this difficulty by explictly bookkeeping all pending constraints. This leads to the following definitions. The game-graph for delayed simulation is $\mathcal{G}^{\mathrm{de}} = \langle V_{\mathrm{E}}^{\mathrm{de}}, V_{\mathrm{A}}^{\mathrm{de}}, \rightarrow^{\mathrm{de}} \rangle$, where nodes in $V_{\mathrm{A}}^{\mathrm{de}}$ are of the form $v_{(\mathbf{q},\mathsf{Bad},\mathbf{s},\mathsf{Good})}$, and nodes in $V_{\mathrm{E}}^{\mathrm{de}}$ of the form $v_{(\mathbf{q},\mathsf{Bad},\mathbf{s},\mathsf{Good},a,\mathbf{q}',\mathbf{s}')}$, with $\mathbf{q}, \mathbf{q}', \mathbf{s}, \mathbf{s}' \subseteq Q$. $\mathsf{Bad} = \langle \mathbf{b}_1 \supset \cdots \supset \mathbf{b}_{m_1} \rangle$ and $\mathsf{Good} = \langle \mathbf{g}_1 \subset \cdots \subset \mathbf{g}_{m_2} \rangle$ are two *sequences* of sets of states from Q, strictly ordered by set-inclusion, which are used to keep track of multiple obligations.

Intuitively, Bad is used to detect when new constraints should be created, i.e., to detect when every *Left* pebble is universally good since some previous round. At each round, a new set of bad pebbles $\mathbf{b} = \mathbf{q} \setminus F$ is added to Bad. When accepting states are visited by *Left* pebbles, they are discarded from every set $\mathbf{b} \in \mathsf{Bad}$. When some $\mathbf{b}$ becomes eventually empty, this means that, at the current round, all *Left* pebbles are universally good since some previous round: At this point, $\mathbf{b}$ is removed from Bad, and we say that *the red light flashes*.

The sequence Good represents a set of constraints to be eventually satisfied. Each $\mathbf{g} \in \mathsf{Good}$ is a set of good pebbles, which we require to "grow" until it becomes equal to $\mathbf{s}$. When $\mathsf{Good} = \emptyset$, there is no pending constraint. Constraints are added to Good when the red light flashes (see above): In this case, we update Good by adding the new empty constraint $\mathbf{g} = \emptyset$. When accepting states are visited by *Right* pebbles, we upgrade every constraint $\mathbf{g} \in \mathsf{Good}$ by adding accepting states. Completed constraints $\mathbf{g} = \mathbf{s}$ are then removed from Good, and we say that *the green light flashes*.

Lemma 7. $\left|V^{\mathrm{de}}\right| \leq 2 \cdot (n+1)^{2(k_1+k_2)} \cdot \left(1 + (k_1+1)^{k_1+1}\right) \cdot \left(1 + 2(k_2+1)^{k_2+1}\right) \cdot |\Sigma|$.

Transitions in $\mathcal{G}^{\mathrm{de}}$ are defined as follows. For any $(\mathbf{q}, \mathbf{s}, a, \mathbf{q}'', \mathbf{s}'') \in \Gamma^{\mathrm{Sp}}$, we have $v_{(\mathbf{q},\mathsf{Bad},\mathbf{s},\mathsf{Good})} \rightarrow^{\mathrm{de}} v_{(\mathbf{q},\mathsf{Bad},\mathbf{s},\mathsf{Good},a,\mathbf{q}'',\mathbf{s}'')}$, and for $(\mathbf{q}, \mathbf{s}, a, \mathbf{q}'', \mathbf{s}'', \mathbf{q}', \mathbf{s}') \in \Gamma^{\mathrm{Dup}}$, we have $v_{(\mathbf{q},\mathsf{Bad},\mathbf{s},\mathsf{Good},a,\mathbf{q}'',\mathbf{s}'')} \rightarrow^{\mathrm{de}} v_{(\mathbf{q}',\mathsf{Bad}',\mathbf{s}',\mathsf{Good}')}$, where $\mathsf{Bad}', \mathsf{Good}'$ are computed according to Algorithm 1 in Appendix E.3 of the technical report [2].

We have that Eve wins iff every red flash is matched by at least one green flash, and different red flashes are matched by different green ones. This can be checked by verifying that infinitely often either $\mathsf{Good} = \emptyset$ or $\mathbf{s} \in \mathsf{Good}$, i.e., it is not the case that Good contains a constraint that it is not eventually "completed" and discarded. Let $T = \{v_{(\mathbf{q},\mathsf{Bad},\mathbf{s},\mathsf{Good})} \mid \mathsf{Good} = \emptyset \vee \mathbf{s} \in \mathsf{Good}\}$, and define the initial configuration as

$$v_I = \begin{cases} v_{(\mathbf{q},\{\mathbf{q}\setminus F\},\mathbf{s},\emptyset)} & \text{if } \mathbf{q} \setminus F \neq \emptyset \\ v_{(\mathbf{q},\emptyset,\mathbf{s},\{\mathbf{s}\cap F\})} & \text{otherwise} \end{cases}$$

$\mathbf{q} \sqsubseteq^{\mathrm{de}}_{k_1,k_2} \mathbf{s}$ iff T is visited infinitely often iff $v_I \in W^{\mathrm{de}} = \nu\mathbf{x}\mu\mathbf{y}\ (\mathrm{cpre}(\mathbf{y}) \cup T \cap \mathrm{cpre}(\mathbf{x}))$.

Theorem 10. *For any fixed $k_1, k_2 \in \mathbb{N}$, $x \in \{\forall\mathrm{di}, \exists\mathrm{di}, \mathrm{de}, \mathrm{f}\}$ and sets $\mathbf{q}, \mathbf{s} \subseteq Q$, deciding whether $\mathbf{q} \sqsubseteq^{x}_{(k_1,k_2)} \mathbf{s}$ can be done in polynomial time.*

7 Conclusions and Future Work

Transitivity for (n, n)-simulations. As discussed at the end of Section 4, composing (n, n) (winning) strategies is apparently much more difficult than in the $(1, n)$ and $(n, 1)$ case. We conjecture that all types of (n, n)-simulations discussed in this paper are transitive, and showing this would conceivably solve the join problem as well.

Quotienting with $(n, 1)$- and (n, n)-simulations. While we have dealt with $(1, n)$-quotients, we have not considered $(n, 1)$- or (n, n)-quotients. For the latter, one should first solve the associated transitivity problem, and, for both, an appropriate notion of semielective-quotient has to be provided. We have shown that this is already a non-trivial task for $(1, n)$-simulations on ABA.

Future directions. Our work on delayed simulation has shown that several generalizations are possible. In particular, two issues need to be addressed. The first is the complexity of the structure of the game-graph needed for computing delayed simulation. A possible generalization of delayed simulation involving looser "synchronization requirements" between obligations and their satisfaction might result in simpler game-graphs. The second issue concerns Lemmas 3 and 4: We would like to find a weaker delayed-like simulation for which the counterexample shown there does not hold. This would give a better understanding of the MH-construction.

As in [4], it is still open to find a hierarchy of (k_1, k_2)-multipebble simulations converging to ω-language inclusion when $k_1 = k_2 = n$.

Acknowledgment. We thank K. Etessami and C. Fritz for helpful discussions, and an anonymous referee for suggesting the comparison to mediated preorder [1].

References

1. Abdulla, P.A., Chen, Y.-F., Holik, L., Vojnar, T.: Mediating for reduction (on minimizing alternating Büchi automata). In: FSTTCS 2009, Dagstuhl, Germany. LIPIcs, vol. 4, pp. 1–12 (2009)
2. Clemente, L., Mayr, R.: Multipebble simulations for alternating automata. Technical Report EDI-INF-RR-1376, University of Edinburgh, School of Informatics (2010), http://www.inf.ed.ac.uk/publications/report/1376.html

3. Etessami, K.: A hierarchy of polynomial-time computable simulations for automata. In: Brim, L., Jančar, P., Křetínský, M., Kučera, A. (eds.) CONCUR 2002. LNCS, vol. 2421, pp. 131–144. Springer, Heidelberg (2002)
4. Etessami, K., Wilke, T., Schuller, R.A.: Fair simulation relations, parity games, and state space reduction for Büchi automata. SIAM J. Comput. 34(5), 1159–1175 (2005)
5. Fritz, C., Wilke, T.: Simulation relations for alternating Büchi automata. Theor. Comput. Sci. 338(1-3), 275–314 (2005)
6. Gurevich, Y., Harrington, L.: Trees, automata, and games. In: STOC 1982: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, pp. 60–65. ACM, New York (1982)
7. Henzinger, T.A., Rajamani, S.: Fair bisimulation. In: Schwartzbach, M.I., Graf, S. (eds.) TACAS 2000. LNCS, vol. 1785, pp. 299–314. Springer, Heidelberg (2000)
8. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
9. Kozen, D.: Results on the propositional mu-calculus. Theor. Comput. Sci. 27, 333–354 (1983)
10. Miyano, S., Hayashi, T.: Alternating finite automata on ω-words. Theoretical Computer Science 32, 321–330 (1984)
11. Vardi, M.Y.: Alternating automata and program verification. In: van Leeuwen, J. (ed.) Computer Science Today. LNCS, vol. 1000, pp. 471–485. Springer, Heidelberg (1995)

Parameterized Verification of Ad Hoc Networks

Giorgio Delzanno[1], Arnaud Sangnier[1], and Gianluigi Zavattaro[2]

[1] University of Genova
[2] University of Bologna

Abstract. We study decision problems for parameterized verification of a formal model of Ad Hoc Networks with selective broadcast and spontaneous movement. The communication topology of a network is represented as a graph. Nodes represent states of individual processes. Adjacent nodes represent single-hop neighbors. Processes are finite state automata that communicate via selective broadcast messages. Reception of a broadcast is restricted to single-hop neighbors. For this model we consider verification problems that can be expressed as reachability of configurations with one node (resp. all nodes) in a certain state from an initial configuration with an arbitrary number of nodes and unknown topology. We draw a complete picture of the decidability boundaries of these problems according to different assumptions on communication graphs, namely static, mobile, and bounded path topology.

1 Introduction

In recent years there has been an increasing interest in the formal specification of protocols used in Ad Hoc Networks. Building on previous models like [7,19,21], in [22] Singh, Ramakrishnan and Smolka define the ω-calculus as a formal model of Ad Hoc Networks with selective broadcast and spontaneous movement. In the ω-calculus a configuration consists of a finite set of processes. Each process has a local state and an interface containing a finite set of group names. A group name represents a possible communication link with other processes in the network. From an abstract point of view, the structure underlying a configuration of the ω-calculus is a finite graph that defines the communication topology of a network. A node in the graph represents the current state of an individual process. There exists and edge between two nodes if the corresponding interfaces share a common group name. Adjacent nodes are called single-hop neighbors. Processes communicate through selective broadcast. Specifically, a broadcast message can be received only by the set of single-hop neighbors of the emitter.

When the number of nodes is fixed a priori, formal models of Ad Hoc Networks like those provided by the ω-calculus can be verified by using finite-state model checking [11] or constraint-based model checking [22]. Lifting the study of verification problems to the parameterized case in which networks have arbitrary size and possibly unknown topology is a challenging problem for this class of distributed systems.

In the present paper we study parameterized verification problems for an automata-based model of Ad Hoc Networks, we named AHN, inspired by the

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 313–327, 2010.

ω-calculus of [22]. Each node of a network is modeled as a finite-state automaton in which local transitions model either an internal action, a broadcast or a reception of a message taken from a finite alphabet. A protocol is defined then as the composition of a finite but arbitrary number of copies of the automaton running in parallel. As in the ω-calculus a configuration can naturally be viewed as a graph that defines the communication topology. We use group names and interfaces to select the set of neighbors of an emitter that are capable to receive a broadcast message. In our model we define verification problems parametric on the size (number of nodes) and shape (topology of the communication graph) of the initial configurations. Our investigations take into account different assumptions on the communication topology. Specifically, we consider configurations either with static or mobile topology and with static and bounded path topology. In the latter case we assume that there is an upper bound on the length of simple paths in the communication graphs underlying the initial configurations. For each of the above mentioned assumptions, we present a systematic analysis of the following decision problems: (COVER) reachability of a configuration with one node in a given state, (TARGET) reachability of a configuration with all nodes in a given state, (REPEAT-COVER) existence of a computation traversing infinitely often configurations with at least one node in a given state.

Our main negative result is that all three parameterized problems are undecidable for arbitrary static topology. The proofs are based on a simulation of a Turing complete formalism which is correct only for topologies of a given size and shape. As the topology is arbitrary, the simulation is preceded by a protocol able to explore the current topology and to start the simulation only if it is of the expected form.

Perhaps surprisingly, all three problems become decidable in the mobile case. This result is similar to what happens in channel systems where introducing lossiness simplifies the verification task [3]. For static bounded path topologies, TARGET and REPEAT-COVER turn out to be undecidable while COVER is still decidable. The latter result is similar to those presented in [23,15] for bounded depth processes with point-to-point communication. However, due to broadcast communication we need to resort to a different proof technique. Namely, even if we use the theory of *well structured transition systems* (WSTS) [1,2,12] as in [15,23], we need to consider a stronger ordering on configurations based on the induced subgraph ordering [5] instead of the subgraph embedding. To the best of our knowledge, this is the first case of application of the induced subgraph ordering in the context of WSTS.

Related Work. Formal models of networks in which all processes receive a broadcast message at once are presented and analyzed in [6,8,19]. In our setting this kind of broadcast is modeled by configurations whose underlying graph is a clique. Selective broadcast has been studied in several process calculi for ad hoc networks and wireless communication like those presented in [7,13,14,16,21], some of which turn out to be extensions of the pi-calculus [17]. A distinguished feature of the ω-calculus [21,22] is that mobility of processes is abstracted from their communication actions, i.e., mobility is spontaneous and it does not involve any communication. In [22] the authors define a constraint-based analysis for configurations with a fixed number of nodes. The shape of topologies leading to bad configurations is constructed in a lazy manner during a symbolic exploration of the state space. The symbolic approach in [22] seems to improve verification results obtained with more standard model checking tools like Uppaal [11]. In the same paper the authors mention that, without name restriction, reachability of a configuration from an initial one is decidable. In the present paper we lift our reachability problems to the parameterized case in which the initial configuration has unknown size and shape. For networks of arbitrary size, in [20] Saksena et al. define a symbolic procedure based on graph-transformations to analyze routing protocol for Ad Hoc Networks. The symbolic representation is based on upward closed sets of graphs ordered by subgraph inclusion. The procedure is not guaranteed to terminate. In our paper we use a different ordering on graphs, namely induced subgraph, for ensuring termination of backward reachability on graphs with paths of bounded length.

Due to lack of space, omitted proofs can be found in [4].

2 A Formal Model for Ad Hoc Network Protocols

Following [22], a configuration of an Ad Hoc Network is modeled as a tuple of nodes $\langle n_1, \ldots, n_k \rangle$ with $k \geq 1$. A node n_i maintains information about the current state of an individual process and its current set of communication links. The behavior of a single node is described by a finite-state automaton, called *process*, in which transitions represent internal, broadcast, or reception actions. Concerning the representation of communication links, a convenient way to describe the network topology is based on the use of group names. Specifically, to each node we associate an interface that defines the set of group names to which the node belongs. Two nodes are connected if their interfaces share at least one common group name. The semantics of the transitions specifies how different nodes interact in a global configuration. We assume that nodes cannot dynamically be created or deleted.

Definition 1. *An Ad Hoc Network Protocol (shortly AHN) is a pair $\langle P, \mathcal{G} \rangle$ where P is a process definition and $\mathcal{G}$ is a denumerable set of group names. A process $P = \langle Q, \Sigma, E, Q_0 \rangle$ defines the behavior of a single node of the network. Here Q is a finite set of control states, Σ is a finite alphabet, $E \subseteq Q \times (\{\tau\} \cup \{\mathbf{b}(a), \mathbf{r}(a) \mid a \in \Sigma\}) \times Q$ is the transition relation, and $Q_0 \subseteq Q$ is a set of initial control states.*

The label τ represents an internal action of a process, the label $\mathbf{b}(a)$ represents the capability of broadcasting message a, and $\mathbf{r}(a)$ the capability of receiving message a.

Definition 2. *Assume $P = \langle Q, \Sigma, E, Q_0 \rangle$. A* node *$n$ is a pair $\langle q, I \rangle$, where $q \in Q$ is called the* state *and $I \subseteq \mathcal{G}$ is the called the* interface *of the node. A*

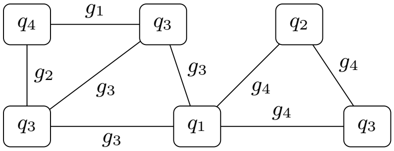


Fig. 1. Graph associated to a configuration

configuration γ *is a tuple* $\langle n_1, \ldots, n_k \rangle$ *of nodes with* size $k \geq 1$. *We use* $\mathcal{C}$ *to denote the set of configurations of any size.*

A configuration γ defines a given network topology specified by the graph $G(\gamma)$. The vertices in $G(\gamma)$ are in bijection with the nodes of γ. The label of a vertex is the state of the corresponding node in γ. Furthermore, there exists an edge between two vertices in $G(\gamma)$ if and only if the intersection of the interfaces of the corresponding nodes in γ is not empty.

For instance, consider a configuration γ with nodes $n_1 = \langle q_4, \{g_1, g_2\}\rangle$, $n_2 = \langle q_3, \{g_1, g_3\}\rangle$, $n_3 = \langle q_3, \{g_2, g_3\}\rangle$, $n_4 = \langle q_1, \{g_3, g_4\}\rangle$, $n_5 = \langle q_2, \{g_4\}\rangle$, and $n_6 = \langle q_3, \{g_4\}\rangle$, the communication topology induced by γ is depicted in Figure 1.

We define functions σ and ι to extract the state and the interface of a node, i.e., $\sigma(\langle q, I\rangle) = q$ and $\iota(\langle q, I\rangle) = I$. We extend σ and ι to configurations in the natural way. For a configuration γ, we sometimes consider $\sigma(\gamma)$ as a set rather than a vector and use $q \in \sigma(\gamma)$ to denote that there exists a node n_i in γ such that $\sigma(n_i) = q$. The set of indexes of nodes adjacent to a node n_i in a configuration $\gamma = \langle n_1, \ldots, n_k\rangle$ (single-hop neighbors of n_i in $G(\gamma)$) is defined as $Shn(\gamma, i) = \{j \in [1..k] \mid \iota(n_i) \cap \iota(n_j) \neq \emptyset \text{ and } j \neq i\}$. For a broadcast message $a \in \Sigma$, we define the set of indexes $Rec(\gamma, a) = \{j \in [1..k] \mid (\sigma(n_j), \mathbf{r}(a), q) \in E$ for some $q \in Q\}$. The set of nodes in γ enabled by a broadcast a sent by node n_i is then defined as $Enabled(\gamma, i, a) = Shn(\gamma, i) \cap Rec(\gamma, a)$.

Operational Semantics. The semantics of an AHN $\langle P = \langle Q, \Sigma, E, Q_0\rangle, \mathcal{G}\rangle$ is given by its associated transition system $TS(P, \mathcal{G}) = \langle \mathcal{C}, \Rightarrow, \mathcal{C}_0\rangle$ (we recall that $\mathcal{C}$ is the set of configurations of all possible size), $\mathcal{C}_0$ is the set of initial configurations defined as $\mathcal{C}_0 = \{\gamma \in \mathcal{C} \mid \sigma(\gamma) \subseteq Q_0\}$ and $\Rightarrow$ is the transition relation in $\mathcal{C} \times \mathcal{C}$ defined as follows.

Definition 3. *For* $\gamma = \langle n_1, \ldots, n_k\rangle$ *and* $\gamma' = \langle n'_1, \ldots, n'_k\rangle$, $\gamma \Rightarrow \gamma'$ *iff one of the following conditions holds:*

Internal. *There exists* $i \in [1..k]$ *such that* $(\sigma(n_i), \tau, \sigma(n'_i)) \in E$, $\iota(n_i) = \iota(n'_i)$, *and* $n'_j = n_j$ *for all* $j \in [1..k] \setminus \{i\}$.

Broadcast. *There exists* $a \in \Sigma$ *and* $i \in [1..k]$ *such that* $(\sigma(n_i), \mathbf{b}(a), \sigma(n'_i)) \in E$, $\iota(n_i) = \iota(n'_i)$, *and the following conditions hold:*
- *For all* $j \in Enabled(\gamma, i, a)$, $(\sigma(n_j), \mathbf{r}(a), \sigma(n'_j)) \in E$ *and* $\iota(n'_j) = \iota(n_j)$;
- *For all* $p \notin (Enabled(\gamma, i, a) \cup \{i\})$, $n'_p = n_p$.

We denote by $\Rightarrow^*$ the reflexive and transitive closure of $\Rightarrow$. An execution is a sequence $\gamma_0\gamma_1 \ldots$ such that $\sigma(\gamma_0) \subseteq Q_0$ and $\gamma_i \Rightarrow \gamma_{i+1}$ for $i \geq 0$.

As an example, consider a process definition in which $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{a\}$, and E contains the rules

$$(q_1, \tau, q_2), \quad (q_2, \mathbf{b}(a), q_4), \quad (q_1, \mathbf{r}(a), q_2), \text{ and } (q_3, \mathbf{r}(a), q_2)\}$$

Starting from a connected component in which nodes have label q_1 or q_3, once an alarm is detected by a q_1 node (τ action), it is flooded (broadcast of message a) to all single-hop neighbors which, in turn, forward the alarm to their neighbors, and so on. After some steps, the alarm reaches all multi-hop neighbors yielding a configuration in which all nodes (in the connected component) have label q_4.

3 Decision Problems

In this section we consider decision problems related to verification of safety and liveness properties like those studied for Petri nets [9,10]. We remark that in our formulation the size and the shape of of the initial configurations is not fixed a priori. In the following definitions we assume an AHN $\langle P, \mathcal{G}\rangle$ with transition system $TS(P, \mathcal{G}) = \langle \mathcal{C}, \Rightarrow, \mathcal{C}_0\rangle$.

The first problem is *control state reachability* (COVER) defined as follows: given a control state q of P, do there exist $\gamma \in \mathcal{C}_0$ and $\gamma' \in \mathcal{C}$ such that $\gamma \Rightarrow^* \gamma'$ and $q \in \sigma(\gamma')$?
We recall that a configuration γ is initial if $\sigma(\gamma) \subseteq Q_0$. Notice that being initial does not enforce any particular constraint on the topology. Thus, assume that the state q represents an error state for a node of the network. If we can solve COVER, then we can decide if there exists a topology of the network and a sufficient number of processes from which we can generate a configuration in which the error is exposed.

The second problem is *target reachability* (TARGET) which we define as follows: given a subset of control states F of P, do there exist $\gamma \in \mathcal{C}_0$ and $\gamma' \in \mathcal{C}$ such that $\gamma \Rightarrow^* \gamma'$ and $\sigma(\gamma') \subseteq F$?
Assume that the subset F represents blocking states for nodes of the network. If we can solve TARGET, then we can decide if there exists a topology of the network and a sufficient number of processes from which we can reach a configuration in which processes can no longer move.

Finally we will also study the *repeated control state reachability problem* (REPEAT-COVER): given a control state q of P, does there exist an infinite execution $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \ldots$ such that the set $\{i \in \mathbb{N} \mid q \in \sigma(\gamma_i)\}$ is infinite?
This problem is a classical extension of the COVER problem that can be used, for instance, to verify whether a protocol is able to react to the occurrence of errors by reaching a state from which errors do not occur any longer. Assume that q represents the error state. If we can solve REPEAT-COVER, then we can decide if there exists a topology of the network and a sufficient number of processes that can generate a computation including infinitely many error states.

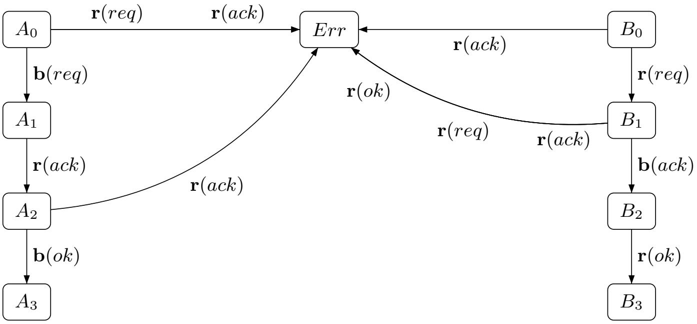


Fig. 2. The RAO (Req/Ack/Ok) protocol

4 Static Topology

In this section, we will prove that COVER, TARGET and REPEAT-COVER are all undecidable problems. We first recall that in our decision problems there are no assumptions on the number of nodes and on the communication topology of the initial configurations. Furthermore, the model does not admit dynamic reconfigurations of the topology. Broadcast communication can be used however to ensure that a specific protocol succeeds only if the network topology has a certain form. To be more precise, consider the protocol specified by the process Req/Ack/Ok (RAO) of Figure 2 where A_0 and B_0 are the initial states. The following property then holds.

Proposition 1. *Let $\mathcal{G}$ be a denumerable set of group names and γ an initial configuration of the AHN $\langle RAO, \mathcal{G}\rangle$. If γ' is a configuration such that $\gamma \Rightarrow^* \gamma'$ and such that $B_3 \in \sigma(\gamma')$, then the graph $G(\gamma')$ has the following properties:*

- *each node n labeled with B_3 is adjacent to a unique node labeled with A_3 (we denote this node by $f(n)$)*[1]*;*
- *for each node n labeled with B_3, all the nodes adjacent to n or $f(n)$ are labeled with Err (except of course n and $f(n)$).*

Proof. Assume n is a node of γ' in state B_3. Since n has received a message *ok* to reach B_3, it is necessarily adjacent to a node in state A_3. No other node adjacent to n can be in state A_3. Indeed, if n receives two *req* messages before sending an *ack*, then n moves to state *Err*. Furthermore, if n sends an *ack*, then all adjacent nodes that are in states A_0 (ready to send a *req*) move to state *Err*. Rule $(A_0, \mathbf{r}(req), Err)$ ensures that, in $G(\gamma')$, no node labeled A_i is adjacent to a node labeled A_3. Rules $(B_0, \mathbf{r}(ack), Err)$ and $(B_1, \mathbf{r}(ack), Err)$ ensure that, when n has label B_3, its single-hop neighbors cannot have label B_i. Rule $(B_1, \mathbf{r}(ok), Err)$ ensures that a node different from n but adjacent to $f(n)$ must have state different from B_i. Indeed, if such a node is in state B_1, then the broadcast *ok* sent by $f(n)$ sends it to *Err*, and if such a node moves to B_2 sending *ack* then it sends node $f(n)$ to *Err* before it can reach A_3. □

[1] Two nodes are adjacent iff there is an edge between them.

Using an extension of the RAO protocol, we can define an AHN which simulates the execution of a deterministic two-counter Minsky machine and reduce the halting problem to COVER. A deterministic Minsky machine manipulates two integer variables c_1 and c_2, which are called counters, and it is composed of a finite set of instructions. Each of the instruction is either of the form (1) $L : c_i := c_i + 1;$ `goto` L' or (2) L : `if` $c_i = 0$ `then goto` L' `else` $c_i := c_i - 1;$ `goto` L'' where $i \in \{1, 2\}$ and L, L', L'' are labels preceding each instruction. Furthermore there is a special label L_F from which nothing can be done. The halting problem consists then in deciding whether or not the execution that starts from L_0 with counters equal to 0 reaches L_F.

The intuition behind the reduction is as follows. In the first phase we use a variant of the RAO protocol to select a control node and two list of nodes, with state representing value 0 or 1, used to simulate the content of the counters. The length of each list must be sufficient to represent the maximum value stored in each counter during the simulation. All other nodes in the vicinity of the control state and of the two lists are sent to an error state during the execution of the first phase. In the second phase the control node starts the simulation of the instructions. It operates by querying and changing the state of the nodes in the two lists according to the type of instructions to be executed. In this phase all nodes in the same list behave in the same way. Requests are propagated back and forth a list by using broadcast sent by a node to its (unique) single-hop successor/predecessor node. The protocols that define the two phases are fairly complicated; the corresponding automata are described in detail in [4]. From the undecidability of the halting problem for two-counter Minsky machines [18], we obtain the following result.

Theorem 1. COVER *is an undecidable problem.*

Furthermore, we have the following corollary.

Corollary 1. TARGET *and* REPEAT-COVER *are undecidable problems.*

Proof. Assume $P = \langle Q, \Sigma, E, Q_0 \rangle$ and let $\mathcal{G}$ be a denumerable set of group names and $q \in Q$. To reduce COVER to REPEAT-COVER we simply add a loop of the form (q, τ, q) to E. To reduce COVER to TARGET, we build the process $P' = \langle Q', \Sigma', E', Q'_0 \rangle$ defined as follows:

- $Q' = Q \uplus \{r_0, r_1, r_F\}$ (with $Q \cap \{r_0, r_1, r_F\} = \emptyset$);
- $\Sigma' = \Sigma \uplus \{F_1, F_2\}$ (with $\Sigma \cap \{F_1, F_2\} = \emptyset$);
- $E' = E \uplus \{(q, \mathbf{b}(F_1), r_F), (r_0, \mathbf{r}(F_1), r_1), (r_1, \mathbf{b}(F_2), r_F)\} \cup \{(q', \mathbf{r}(F_2), r_F) \mid q' \in Q\}$;
- $Q'_0 = Q_0 \uplus \{r_0\}$.

Let $TS(P, \mathcal{G}) = \langle \mathcal{C}, \Rightarrow, \mathcal{C}_0 \rangle$ and $TS(P', \mathcal{G}) = \langle \mathcal{C}', \Rightarrow', \mathcal{C}'_0 \rangle$. It is then easy to see that there exist $\gamma_2 \in \mathcal{C}'_0$ and $\gamma'_2 \in \mathcal{C}'$ such that $\gamma_2 \Rightarrow'^{*} \gamma'_2$ and $\sigma(\gamma'_2) \subseteq \{r_F\}$ if and only if there exists $\gamma_1 \in \mathcal{C}_0$ and $\gamma'_1 \in \mathcal{C}$ such that $\gamma_1 \Rightarrow^{*} \gamma'_1$ and $q \in \sigma(\gamma'_1)$. In fact, in $TS(P', \mathcal{G})$ after being in the state q a node can broadcast the message F_1 which launches a protocol whose goal is to send all the other nodes in the state r_F. □

5 Mobile Topology

In this section we consider a variant on the semantics of AHN obtained by adding spontaneous movement of nodes as in [22]. Node mobility is modeled by non-deterministic updates of their interfaces. Formally, let $\langle P, \mathcal{G} \rangle$ be a AHN with $TS(P, \mathcal{G}) = \langle \mathcal{C}, \Rightarrow, \mathcal{C}_0 \rangle$. The semantics of $\langle P, \mathcal{G} \rangle$ with mobility is given by the transition system $TS_M(P, \mathcal{G}) = \langle \mathcal{C}, \Rightarrow_M, \mathcal{C}_0 \rangle$ where the transition $\Rightarrow_M$ is defined as follows.

Definition 4 (Transition Relation with Mobility). *For $\gamma, \gamma' \in \mathcal{C}$ with $\gamma = \langle n_1, \ldots, n_k \rangle$ and $\gamma' = \langle n'_1, \ldots, n'_k \rangle$, we have $\gamma \Rightarrow_M \gamma'$ iff one the following conditions holds:*

- *$\gamma \Rightarrow \gamma'$ (**no movement**);*
- *there exists $i \in [1..k]$ such that, $\sigma(n'_i) = \sigma(n_i)$ (state does not change), $\iota(n'_i) \subseteq \mathcal{G}$ (interface changes in an arbitrary way), and for all $j \in [1..k] \setminus \{i\}$, $n'_j = n_j$ (all other nodes remain unchanged) (**movement**).*

We prove next that COVER, REPEAT-COVER and TARGET are decidable for AHN with mobility. Intuitively, this follows from the observation that the topology of the network changes in an unpredictable and uncontrollable manner. Hence, every broadcast message sent by a node is received by a non-deterministically chosen set of nodes, namely those in the transmission range of the emitter at the time the message is sent. Formally, we reduce COVER, TARGET and REPEAT-COVER respectively to the *marking coverability*, *marking reachability* and *repeated marking coverability* problems for Petri nets, which are known to be decidable [9,10].

A *Petri net* (see e.g. [9]) is a tuple $N = (S, T, m_0)$, where S and T are finite sets of *places* and *transitions*, respectively. A finite multiset over the set S of places is called a *marking*, and m_0 is the initial marking. Given a marking m and a place p, we say that the place p contains $m(p)$ *tokens* in the marking m if there are $m(p)$ occurrences of p in the multiset m. A transition is a pair of markings written in the form $m' \mapsto m''$. The marking m of a Petri net can be modified by means of transitions firing: a transition $m' \mapsto m''$ can fire if $m(p) \geq m'(p)$ for every place $p \in S$; upon transition firing the new marking of the net becomes $n = (m \setminus m') \uplus m''$ where $\setminus$ and $\uplus$ are the difference and union operators for multisets, respectively. This is written as $m \rightarrow n$. We use $\rightarrow^*$ [resp. $\rightarrow^+$] to denote the reflexive and transitive closure [resp. the transitive closure] of $\rightarrow$. We say that m' is *reachable from* m if $m \rightarrow^* m'$. The *coverability* problem for marking m consists of checking whether $m_0 \rightarrow^* m'$ with $m'(p) \geq m(p)$ for every place $p \in S$. The *reachability* problem for marking m consists of checking whether $m_0 \rightarrow^* m$. Finally, the *repeated coverability problem* for marking m consists of checking wether there exists an infinite execution $m_0 \rightarrow^+ m_1 \rightarrow^+ m_2 \rightarrow^+ \ldots$ such that for all $i \in \mathbb{N}$, $m_i(p) \geq m(p)$ for every place $p \in S$. The coverability, reachability and repeated coverability problems are decidable for Petri nets [9,10].

We now show how to build a Petri net which simulates the behavior of an AHN with mobility. Figure 3 gives an example of a Petri net associated to a process. In the Petri net, each control state q has a corresponding place q, and

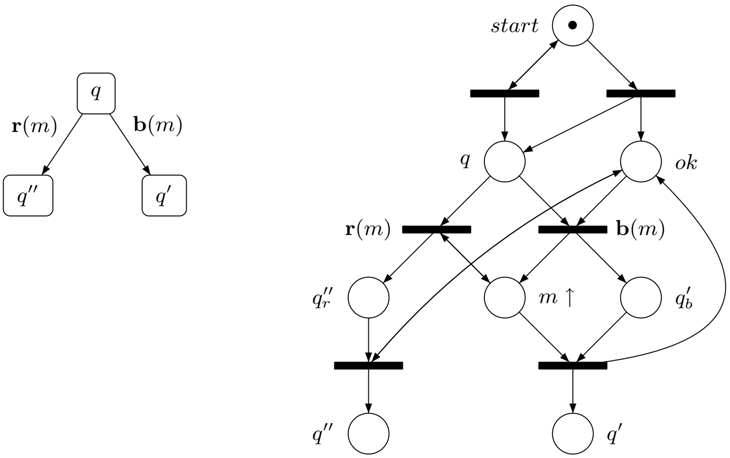


Fig. 3. A Petri net which simulates an AHN with mobility

each node $\langle q, I \rangle$ of the network is represented by a token in the place q. The interfaces of nodes (thus also the network topology) are abstracted away in the Petri net. In the first phase, the net non-deterministically puts tokens in the places corresponding to the initial control states of the process. Then it produces a token in the place *ok* and the simulation begins. The broadcast communication is modeled by a *broadcast protocol* whose effect is to deliver the emitted message to a non-deterministically chosen set of potential receivers. More precisely, the broadcast protocol can be started by a token in a place q such that $(q, \mathbf{b}(m), q')$; then the token is moved to a transient place q'_b and a token is produced in the place $m \uparrow$. During the execution of the protocol, every token in a place r such that $(r, \mathbf{r}(m), r')$ can receive the message moving in a transient place r'_r. The protocol ends when the token in the transient place q'_b moves to the place q'. The tokens in the transient places r'_r can move to the corresponding places r' only when no broadcast protocol is running (when a broadcast protocol is running, there is no token in the place *ok*). This broadcast protocol does not faithfully reproduce the broadcast as formalized in the AHN model: in fact, in the Petri net there is no guarantee that the tokens in the transient places r'_r move to the corresponding places r' at the end of the execution of the protocol. A token that remains in those transient places (thus losing the possibility to interact with the other tokens in the Petri net) corresponds to a node in the AHN model that disconnects, due to mobility, from the other nodes in the system. Testing whether there is an execution in the AHN with mobility which ends in a configuration where one of the nodes is in the control state q can be done by testing whether the marking $\{q, ok\}$ can be covered in the associated Petri net. Hence:

Theorem 2. *There exists a reduction from the* COVER *problem for AHN with mobility to the marking coverability problem for Petri nets.*

Using the same construction, we also obtain:

Theorem 3. *There exists a reduction from the* REPEAT-COVER *problem for AHN with mobility to the marking repeated coverability problem for Petri nets.*

In order to reduce TARGET to marking reachability we need to extend the Petri net associated to an AHN, adding a final stage in the computation, dedicated to the elimination of tokens from the places corresponding to the final states in F. Intuitively we add a transition of the form $\{ok\} \mapsto \{end\}$ and for each $q \in F$ we add a transition $\{end, q\} \mapsto \{end\}$ and we then test if the marking where all the places are empty except the place *end* is reachable.

Theorem 4. *There exists a reduction from the* TARGET *problem for AHN with mobility to the marking reachability problem for Petri nets.*

From these three last theorems and from the fact that the marking coverability, marking repeated coverability and marking reachability problems are decidable for Petri nets, we finally deduce:

Corollary 2. COVER*,* REPEAT-COVER *and* TARGET *are decidable for AHN with mobility.*

6 Static Bounded Path Topology

Let us go back to the AHN model with static topology. The possibility for a message to pass through an unbounded number of new nodes is a key feature in the proof of Theorem 1 (undecidability of COVER for static topology). For this reason, it seems natural to study COVER, TARGET and REPEAT-COVER for a restricted class of configurations in which, for a fixed K, a message can pass through at most K-different nodes. Formally, given an AHN $\langle P, \mathcal{G} \rangle$ with $TS(P, \mathcal{G}) = \langle \mathcal{C}, \Rightarrow, \mathcal{C}_0 \rangle$ our class of restricted configurations is defined as follows:

Definition 5. *Given an integer $K \geq 1$, a configuration γ is a K-*bounded path configuration *if the longest simple path in the associated graph $G(\gamma)$ has length at most K.*

We denote by $\mathcal{C}^K$ the set of K-bounded path configurations. The semantics of the AHN $\langle P, \mathcal{G} \rangle$ resticted to K-bounded path configurations is given by the transition system $TS_K(P, \mathcal{G}) = \langle \mathcal{C}^K, \Rightarrow_K, \mathcal{C}_0^K \rangle$ where the transition relation $\Rightarrow_K$ is the restriction of $\Rightarrow$ to $\mathcal{C}^K \times \mathcal{C}^K$ and $\mathcal{C}_0^K = \mathcal{C}_0 \cap \mathcal{C}^K$. For fixed K, the class of K-bounded path configurations contains an infinite set of graphs. To give some examples, stars with a center node and any number of satellite nodes have always 3-bounded paths, whereas cliques of arbitrary size may have paths of unbounded length.

6.1 Decidability of COVER

In order to study COVER restricted to bounded path configurations, we first introduce some definitions and prove auxiliary properties. First of all, we give the definition of the *induced subgraph* relation.

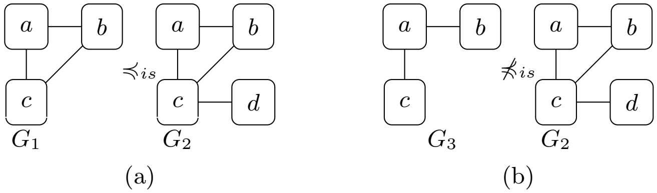


Fig. 4. Examples of the induce subgraph relation

Definition 6. *For configurations γ_1 and γ_2, we define $\gamma_1 \preccurlyeq_{is} \gamma_2$ if there exists a label preserving injection h from nodes of $G_1 = G(\gamma_1)$ to nodes of $G_2 = G(\gamma_2)$ such that (n, n') is an edge in G_1* if and only if *$(h(n), h(n'))$ is an edge in G_2, i.e., G_1 is* isomorphic *to an induced subgraph of G_2.*

Notice that the induced subgraph relation is stronger than the *subgraph relation.* The subgraph ordering requires only a homomorphic embedding of G_1 into G_2. To illustrate, in Fig. 4 (a) G_1 is isomorphic to an induced subgraph of G_2, hence $G_1 \preccurlyeq_{is} G_2$. In (b) G_3 is obtained from G_1 by removing the edge from node a to node c. The induced graph of G_2 with nodes a, b, c is no more isomorphic to G_3, hence $G_3 \not\preccurlyeq_{is} G_2$. Notice, however, that G_3 is still a subgraph of G_2. The following lemma then holds.

Lemma 1. *Given $K \geq 1$, $(\mathcal{C}^K, \preccurlyeq_{is})$ is a well-quasi ordering (shortly wqo), i.e., for every infinite sequence of K-bounded path configurations $\gamma_1\gamma_2\ldots$ there exist $i < j$ s.t. $\gamma_i \preccurlyeq_{is} \gamma_j$.*

Proof. It immediately follows from Ding's Theorem (Theorem 2.2 in [5]).

Given a subset $S \subseteq \mathcal{C}^K$ we define $S\uparrow = \{\gamma' \in \mathcal{C}^K \mid \gamma \in S$ and $\gamma \preccurlyeq_{is} \gamma'\}$, i.e., $S\uparrow$ is the set of configurations generated by those in S via $\preccurlyeq_{is}$. A set $S \subseteq \mathcal{C}^K$ is an *upward closed set* w.r.t. to $(\mathcal{C}^K, \preccurlyeq_{is})$ if $S\uparrow = S$. Since $(\mathcal{C}^K, \preccurlyeq_{is})$ is a wqo, we obtain that every set of configurations that is upward closed w.r.t. $(\mathcal{C}^K, \preccurlyeq_{is})$ has a finite basis, i.e., it can be finitely represented by a finite number of K-bounded path configurations. We can exploit this property to define a decision procedure for COVER. For this purpose, we apply the methodology proposed in [1]. The first property we need to prove is that the transition relation induced by our model is compatible with $\preccurlyeq_{is}$.

Lemma 2 (Monotonicity). *For every $\gamma_1, \gamma_2, \gamma_1' \in \mathcal{C}^K$ such that $\gamma_1 \Rightarrow_K \gamma_2$ and $\gamma_1 \preccurlyeq_{is} \gamma_1'$, there exists $\gamma_2' \in \mathcal{C}^K$ such that $\gamma_1' \Rightarrow_K \gamma_2'$ and $\gamma_2 \preccurlyeq_{is} \gamma_2'$.*

Proof. For lack of space, we focus here on the application of a broadcast rule with label $\mathbf{b}(a)$. Assume that the rule is applied to a node n adjacent in $G(\gamma_1)$ to nodes $N = \{n_1, \ldots, n_k\}$. Assume that the subset N' of N contains nodes that are enabled by message a. By applying the operational semantics, the state of n and the states of nodes in N' are updated simultaneously Assume now that $G(\gamma_1)$ is isomorphic to an induced subgraph of $G(\gamma_1')$ via the injection h. Then,

$h(n)$ is adjacent to the set of nodes $h(N)$ (there cannot be more connections since $h(G(\gamma_1))$ is an induced subgraph of $G(\gamma_1'))$. Thus, the same rule is enabled in $h(n)$ and in $h(N')$ and yields the same effect on the labels. Thus, we obtain γ_2' such that $G(\gamma_2) \preccurlyeq_{is} G(\gamma_2')$. □

Monotonicity ensures that if S is an upward closed set of configurations (w.r.t. $(\mathcal{C}^K, \preccurlyeq_{is})$), then the set of predecessors of S accroding to $\Rightarrow_K$, defined as $\mathit{pre}_K(S) = \{\gamma \mid \gamma \Rightarrow_K \gamma' \text{ and } \gamma' \in S\}$, is still upward closed. We now show that we can effectively compute a finite representation of $S \cup \mathit{pre}_K(S)$.

Lemma 3. *Given a finite basis B of an upward closed set $S \subseteq \mathcal{C}^K$, there exists an algorithm to compute a finite basis B' of $S \cup \mathit{pre}_K(S)$ s.t. $S \cup \mathit{pre}_K(S) = B' \uparrow$.*

Proof. We focus on the the backward application of a broadcast rule $(q, \mathbf{b}(a), q')$ to a configuration in the upward closure of B. The computation of the set $B \uparrow \cup\, \mathit{pre}_K(B \uparrow)$, where $\mathit{pre}_K(B \uparrow) = \{\gamma \mid \gamma \Rightarrow_K \gamma' \text{ and } \gamma' \in B \uparrow\}$ is done according to the following steps.
Initially, we set $B' := B$.
Then, for each $\gamma \in B$:

1. For each vertex n labeled with q' in the graph $G(\gamma)$, let N be the set of nodes adjacent to q', we apply then the following rule:
   - If there exists a node in N with state r such that $(r, \mathbf{r}(a), r')$ is a rule in the model, then we add no predecessor to B' (because every node $n' \in S$ in state r **must** react to the broadcast);
   - otherwise, for any subset $N' = \{n_1, \ldots, n_k\}$ of nodes in N such that n_i has state r_i' and $(r_i, \mathbf{r}(a), r_i')$ is a rule in the model, we build a predecessor configuration γ' in which the label of n is updated to q and the label of n_i is updated to r_i for $i \in \{1, \ldots, k\}$ and if there is no γ'' in B' such that $\gamma'' \preccurlyeq_{is} \gamma'$, we add γ' to B' (Note that we have to select all possible subset N' of N because we must consider the cases in which nodes connected to n are already in the target state of some reception rule).
2. Let Γ' be the set of configurations γ' in $\mathcal{C}^K$ obtained by adding a node n in state q' to γ such that in $G(\gamma')$, n is adjacent to at least one node (i.e. in Γ' we have all the configurations obtained by added a connected node to γ and which are still K-bounded path configurations). We then apply the precedent rule 1. to each configuration in Γ' considering the added node n labeled with q'. □

We can now state the main theorem of this section.

Theorem 5. *For $K \geq 1$,* COVER *is decidable for AHN restricted to K-bounded path configurations.*

Proof. From Lemmas 1, 2, and 3 it follows that the transition system induced by any AHN is well structured with respect to $(\mathcal{C}^K, \preccurlyeq_{is})$. The theorem then follows from the general properties of well structured transition systems [1,2,12]. The decision algorithm is based on a symbolic backward exploration of the state space that, starting from a graph with a single node denoting the target state, saturates the set of symbolic predecessors computed using the operator described

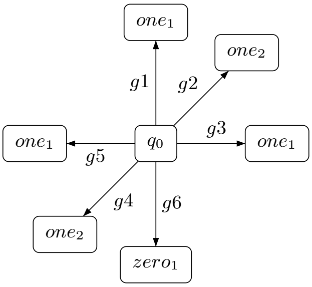


Fig. 5. Configuration $\langle q_0, c_1 = 3, c_2 = 2\rangle$ for $c_1 \in [0, 4]$ and $c_2 \in [0, 2]$

in Lemma 3. Termination is ensured by the wqo property of the induced subgraph relation over K-bounded path configurations. □

6.2 Undecidability of TARGET and REPEAT-COVER

In order to show that TARGET is undecidable for K-bounded path configurations, we show how to model a Minsky machine in such a way that the machine terminates if and only if the corresponding AHN has a computation (restricted to K-bounded path configurations) that reaches a configuration in which all nodes are in some specific final state. For this purpose, we design a protocol that succeeds only on star topologies in which the center node represents the current control state and the satellite nodes the units of the two counters. Such units are initially in the $zero_i$ state (with $i \in \{1, 2\}$). The number of satellite nodes needed to guess the maximal values reached by the counters during the computation is non-deterministically chosen in a preliminary part of the simulation. Only runs that initially guess a sufficient number of satellite nodes can successfully terminate the simulation. A satellite node moves from the $zero_i$ to the one_i state when the i-th counter is incremented, and a single node moves from the one_i back to the $zero_i$ state when the counter is decremented. For instance, the star in Figure 5 represents a configuration with control state q_0 and counters $c_1 = 3$ (with maximal value equals to 4), and $c_2 = 2$ (with maximal value equals to 2).

The simulation of instructions with zero-test is a more difficult task. The problem is that it is not possible to check the absence of neighbors with state one_i. Nevertheless, it is possible to ensure that no node is in the state one_i after a test for zero is executed. It is sufficient to use broadcast communication to move all the satellite nodes in the one_i state to a special *sink* state. If the simulation terminates exposing the final control state and no node is in the *sink* state (i.e. a configuration is reached in which all the nodes are in the final control state, in the $zero_i$, or the one_i state), we can conclude that the simulated computation is correct, thus also the corresponding Minsky machine terminates.

Note that the number of satellite nodes is not fixed a priori. However the graph have bounded path (the construction works for paths of length 3), so we can conclude what follows:

Theorem 6. TARGET *is undecidable for AHN restricted to K-bounded path configurations (with $K \geq 3$).*

As a corollary we now prove the undecidability of REPEAT-COVER. We need to slightly modify the way we model Minsky machines. The idea is to repeatedly simulate the computation of the Minsky machine, in such a way that the final control state can be exposed infinitely often if and only if the simulated Minsky machine terminates.

Every simulation phase simulates only a finite number of steps of the Minsky machine, and if the final control state is reached then a new simulation phase is started. This is achieved by including in the initial star topology also satellite nodes in the $free$ state, and ensuring that every simulated action moves one of those nodes to the *done* state. In this way, a simulation cannot perform more steps than the number of $free$ nodes in the initial star topology. If the final control state is reached, a new simulation is started by moving all the nodes from the *done* to the $free$ state, all the nodes from the one_i to the $zero_i$ state, and by restarting from the initial control state. Notice that nodes reaching the *sink* state (due to a wrong execution of a test for zero action) are no longer used in the computation. For this reason, as every time a wrong test for zero is executed some node moves in the *sink* state, we are sure that only finitely many wrong actions can occur. Hence, if the final control state is exposed infinitely often, we have that only finitely many simulation phases could be wrong, while infinitely many are correct. As all simulation phases reach the final control state (necessary to start the subsequent phase), we have that the corresponding Minsky machine terminates. Hence, we have the following Corollary of Theorem 6:

Corollary 3. REPEAT-COVER *is undecidable for AHN restricted to K-bounded path configurations (with $K \geq 3$).*

7 Conclusions

In this paper we have studied different types of verification problems for a formal model of Ad Hoc Networks in which communication is achieved via a selective type of broadcast. Perhaps surprisingly, a model with static topology turns out to be more difficult to analyze with respect to a model with spontaneous node movement. A similar dichotomy appears in verification of perfect and lossy channel systems. Studying the expressiveness of other variations on the semantics to model for instance conflicts, noise and lossiness, is an interesting research direction for future works.

Acknowledgments. The authors would like to thank Prof. Barbara Koenig for having pointed out to us the connection with bounded-depth processes, and Prof. Guoli Ding for useful insights into the theory of graph orderings.

References

1. Abdulla, P.A., Čerāns, C., Jonsson, B., Tsay, Y.-K.: General decidability theorems for infinite-state systems. In: LICS 1996, pp. 313–321 (1996)
2. Abdulla, P.A., Čerāns, C., Jonsson, B., Tsay, Y.-K.: Algorithmic analysis of programs with well quasi-ordered domains. Inf. Comput. 160(1-2), 109–127 (2000)

3. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Inf. Comput. 127(2), 91–101 (1996)
4. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized verification of Ad Hoc Networks (Extended version). DISI-TR-10-01 (June 2010), `http://www.disi.unige.it/index.php?research/techrep`
5. Ding, G.: Subgraphs and well quasi ordering. J. of Graph Theory 16(5), 489–502 (1992)
6. Emerson, E.A., Namjoshi, K.S.: On model checking for non-deterministic infinite-state systems. In: LICS 1998, pp. 70–80 (1998)
7. Ene, C., Muntean, T.: A broadcast based calculus for Communicating Systems. In: IPDPS 2001, p. 149 (2001)
8. Esparza, J., Finkel, A., Mayr, R.: On the verification of Broadcast Protocols. In: LICS 1999, pp. 352–359 (1999)
9. Esparza, J., Nielsen, M.: Decidability issues for Petri nets - a survey. Bulletin of the EATCS 52, 245–262 (1994)
10. Esparza, J.: Some applications of Petri Nets to the analysis of parameterised systems. Talk at WISP 2003 (2003)
11. Fehnker, A., van Hoesel, L., Mader, A.: Modelling and verification of the LMAC protocol for wireless sensor networks. In: Davies, J., Gibbons, J. (eds.) IFM 2007. LNCS, vol. 4591, pp. 253–272. Springer, Heidelberg (2007)
12. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! TCS 256(1-2), 63–92 (2001)
13. Godskesen, J.C.: A calculus for Mobile Ad Hoc Networks. In: Murphy, A.L., Vitek, J. (eds.) COORDINATION 2007. LNCS, vol. 4467, pp. 132–150. Springer, Heidelberg (2007)
14. Merro, M.: An observational theory for Mobile Ad Hoc Networks. Inf. Comput. 207(2), 194–208 (2009)
15. Meyer, R.: On boundedness in depth in the pi-calculus. In: IFIP TCS 2008, pp. 477–489 (2008)
16. Mezzetti, N., Sangiorgi, D.: Towards a calculus for wireless systems. ENTCS 158, 331–353 (2006)
17. Milner, R.: Communicating and mobile systems: the pi-calculus. Cambridge Univ. Press, Cambridge (1999)
18. Minsky, M.: Computation: finite and infinite machines. Prentice Hall, Englewood Cliffs (1967)
19. Prasad, K.V.S.: A Calculus of Broadcasting Systems. Sci. of Comp. Prog. 25(2-3), 285–327 (1995)
20. Saksena, M., Wibling, O., Jonsson, B.: Graph grammar modeling and verification of Ad Hoc Routing Protocols. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 18–32. Springer, Heidelberg (2008)
21. Singh, A., Ramakrishnan, C.R., Smolka, S.A.: A process calculus for Mobile Ad Hoc Networks. In: Lea, D., Zavattaro, G. (eds.) COORDINATION 2008. LNCS, vol. 5052, pp. 296–314. Springer, Heidelberg (2008)
22. Singh, A., Ramakrishnan, C.R., Smolka, S.A.: Query-Based model checking of Ad Hoc Network Protocols. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 603–661. Springer, Heidelberg (2009)
23. Wies, T., Zufferey, D., Henzinger, T.A.: Forward analysis of depth-bounded processes. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 94–108. Springer, Heidelberg (2010)

Termination in Impure Concurrent Languages

Romain Demangeon[1], Daniel Hirschkoff[1], and Davide Sangiorgi[2]

[1] ENS Lyon, Université de Lyon, CNRS, INRIA, France
[2] INRIA/Università di Bologna, Italy

Abstract. An impure language is one that combines functional and imperative constructs. We propose a method for ensuring termination of impure concurrent languages that makes it possible to combine term rewriting measure-based techniques with traditional approaches for termination in functional languages such as logical relations. The method can be made parametric on the termination technique employed on the functional part; it can also be iterated.

We illustrate the method in the case of a π-calculus with both functional and imperative names, and show that, with respect to previous approaches to termination, it allows us to extend considerably the set of processes that can be handled.

The method can also be applied to sequential languages, e.g., λ-calculi with references.

1 Introduction

In this paper, an impure language is one that combines functional and imperative constructs; for instance, a λ-calculus with references, or a π-calculus with functional names.

The π-calculus is naturally imperative. A π-calculus name may however be considered functional if it offers a service immutable over time and always available. Special laws, such as copying and parallelisation laws, are only valid for functional names. As a consequence, the distinction between functional and imperative names is often made in applications. Examples are intermediate target languages of compilers of programming languages based on π-calculus, such as Pict [PT00], in which the peculiarities of functional names are exploited in code optimisations. In the π-calculus language in this paper, the distinction between functional and imperative names is made directly in the syntax. The functional names are introduced with a dedicated construct akin to the 'letrec' of functional languages. The other names — the 'ordinary' π-calculus names — are called imperative. Correspondingly, a process is functional or imperative if it uses only functional or only imperative names.

The subject of this paper is termination in impure concurrent languages. There are powerful techniques that ensure termination in purely functional languages, notably realisability and logical relations, that exploit an assignment of types to terms. Attempts at transporting logical relations onto concurrent languages have had limited success. In the π-calculus, the sets of processes which

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 328–342, 2010.

have been proved to be terminating with logical relations [YBH04, San06] are mainly "functional": they include the standard encodings of the λ-calculus into the π-calculus, but fail to capture common patterns of usage of imperative names.

The other kind of techniques for termination proposed for concurrent languages such as π-calculus uses ideas from term-rewriting systems, with a well-founded measure that decreases as the computation proceeds [DS06]. Such measure-techniques are suitable to handling imperative languages and correspondingly, in the π-calculus, imperative processes. In contrast, the techniques can handle only a limited class of functions, or functional processes. In languages richer than (the analogue of) simply-typed λ-calculi, for instance, measure-based proofs may be impossible (e.g., no measure argument representable in 2nd-order arithmetic could prove termination of the polymorphic λ-calculus, as this would amount to proving consistency of the arithmetic; termination can however be proved with logical relations). In this paper we propose a method for the termination of concurrent languages that combines the measure-based techniques of imperative languages and the logical relation techniques of functional languages. The method can be made parametric on the functional sublanguage and its termination proof.

We illustrate the combination of the two kinds of techniques from their simplest instance, in which the types used in logical relations are those of a simply-typed calculus, and the measure needed in the term-rewriting technique strictly decreases at every imperative computation step.

We explain the basic idea of the method. We first extend the imperative measure to the functional constructs, but without the ambition of ensuring termination: the control on the measure performed on functional constructs is looser than the one performed on the imperative ones. Therefore, while the overall measure of a term decreases along imperative reductions (i.e., steps stemming from imperative constructs), the measure may increase along the functional ones. Indeed the measure assignment also captures divergent terms. However, termination holds if the purely functional sublanguage is terminating. The termination of this sublanguage can be established separately. The soundness proof of the method proceeds by contradiction. The crux is to prune a divergence of an impure term into a divergence of a purely functional term.

The termination condition for the functional subcalculus (the simply-typed system) can be combined with the measure assignment. The result is a type system that refines the simply-typed one by attaching a measure to each type. The final termination result can therefore be stated as a well-typedness condition in this type system.

The method subsumes both component techniques, in the sense that the resulting terminating terms include those that can be proved terminating with logical relations or measures alone. For instance, a simply-typed purely functional term is accepted because no measure-based constraint is generated.

In the soundness proof of the method we never have to manipulate logical relations (that are used to establish termination for the functional sublanguage). We indeed take the functional sublanguage and its termination proof as a black box that we insert into our soundness proof. We can therefore generalise the method and make it parametric with respect to the functional core and the termination technique employed on it. The method could also be iterated, that is, applied more than once (in this case all iterations but the last one are applied on a purely functional language, a subset of which is treated using measures, another subset with other means).

Further, we show that the method can be enhanced by building it on top of more sophisticated measure-based techniques.

The method can also be applied to impure *sequential* languages, e.g., λ-calculi with references. While the schema of the method is the same, the technical details, in particular the key step of pruning and the definition of stratification, are quite different, and will be presented elsewhere (the interested reader is referred to [DHS10b]).

We mostly give only proof sketches of important results, for lack of space. Details and additional material can be found in [DHS10a].

2 An Impure π-Calculus

The π-calculus, in its standard presentation, is imperative: the service offered by a name (its input end) may change over time; it may even disappear. There may be however names whose input end occurs only once, is replicated, and is immediately available. This guarantees that all messages sent along the name will be consumed, and that the continuation applied to each such message is always the same. In the literature these names are called *functional*, and identified either by the syntax, or by a type system. The remaining names are called *imperative*.

In the π-calculus we use, functional names are introduced using a `def` construct, akin to a "letrec" (as said above, we could as well have distinguished them using types). We use a, b for imperative names, f, g for functional names, x, y, c to range over both categories, and v, w for values; a value can be a name or $\star$ (unit). In examples, later, the grammar for values may be richer – integers, tuples, etc.; such additions are straightforward but would make the notations heavier.

$$\begin{array}{lrl}
\text{(Processes)} & P ::= & P_1|P_2 \;\big|\; \mathbf{0} \;\big|\; \overline{c}\langle v\rangle.P \;\big|\; \texttt{def}\ f = (x).P_1\ \texttt{in}\ P_2 \\
 & & \big|\; (\boldsymbol{\nu} a)\,P \;\big|\; c(x).P \;\big|\; !c(x).P \\
\text{(Values)} & v ::= & a \;\big|\; f \;\big|\; \star
\end{array}$$

We sometimes use the CCS notation ($a.P$, $\overline{c}.P$, $\texttt{def}\ f = P_1\ \texttt{in}\ P_2, \dots$) for inputs and outputs that carry the unit value $\star$; we omit trailing occurrences of $\mathbf{0}$ under a prefix. We write $\mathrm{fn}(P)$ for the set of free names of P (the binding constructs are restriction, input and `def`, the latter binding both the functional channel – f in the grammar above – and the received name – x).

An *input-unguarded output* in P is an output prefix that does occur (i) neither under an input in P (ii), nor, for any subterm of P of the form $\texttt{def}\ f = (x).P_1\ \texttt{in}\ P_2$, in the P_1 subterm.

$$(\textbf{call})\ \frac{\mathrm{capt}(\mathbf{E}_2) \cap \mathrm{fn}((x).P) = \emptyset}{\begin{array}{l}\mathbf{E}_1[\texttt{def}\ f = (x).P\ \texttt{in}\ \mathbf{E}_2[\overline{f}\langle v\rangle.Q]] \\ \longrightarrow \mathbf{E}_1[\texttt{def}\ f = (x).P\ \texttt{in}\ \mathbf{E}_2[P\{v/x\} \mid Q]]\end{array}}$$

$$(\textbf{trig})\ \frac{}{\mathbf{E}[\overline{a}\langle v\rangle.Q \mid !a(x).P] \longrightarrow \mathbf{E}[Q \mid P\{v/x\} \mid !a(x).P]}$$

$$(\textbf{comm})\ \frac{}{\mathbf{E}[\overline{a}\langle v\rangle.Q \mid a(x).P] \longrightarrow \mathbf{E}[Q \mid P\{v/x\}]}$$

$$(\textbf{cong})\ \frac{Q \equiv P \qquad P \longrightarrow P' \qquad P' \equiv Q'}{Q \longrightarrow Q'}$$

Fig. 1. Reduction for $\pi_{\texttt{ST}}$

We call $\pi_{\texttt{ST}}$ the simply-typed version of this calculus (the discipline of simple types in π is standard – it is intrinsically contained in the type system we present in Section 3). Typing in particular ensures that the subject c of inputs $c(x).P$ and $!c(x).P$ is always an imperative name.

Evaluation contexts are given by the following grammar:

$$\mathbf{E} = [\,] \;\big|\; \mathbf{E}|P \;\big|\; (\boldsymbol{\nu} a)\,\mathbf{E} \;\big|\; \texttt{def}\ f = (x).P\ \texttt{in}\ \mathbf{E}$$

We write $\mathbf{E}[P]$ for the process obtained by replacing the hole $[\,]$ in $\mathbf{E}$ with P, and $\mathrm{capt}(\mathbf{E})$ stands for the *captured names* of $\mathbf{E}$, that is, the names which are bound at the occurrence of the hole in $\mathbf{E}$.

Structural congruence between processes, written $\equiv$, is defined as the least congruence that is an equivalence relation, includes α-equivalence, satisfies the laws of an abelian monoid for $|$ (with $\mathbf{0}$ as neutral element), and satisfies the following two extrusion laws:

$$P \,|\, (\boldsymbol{\nu} a)\, Q \;\equiv\; (\boldsymbol{\nu} a)\,(P|Q) \qquad \text{if } a \notin \mathrm{fn}(P)$$

$$P \,|\, \texttt{def}\ f = (x).Q_1\ \texttt{in}\ Q_2 \equiv\ \texttt{def}\ f = (x).Q_1\ \texttt{in}\ P|Q_2 \qquad \text{if } f \notin \mathrm{fn}(P).$$

(Rules for replication or for swapping consecutive restrictions or definitions are not needed in $\equiv$.) We extend $\equiv$ to evaluation contexts.

Figure 1 presents the rules defining the reduction relation on $\pi_{\texttt{ST}}$ processes, written $\longrightarrow$. $P\{v/x\}$ stands for the result of the application of the (capture avoiding) substitution of x with v in P. In rule (**call**), we impose that the intrusion of $P\{v/x\}$ in the context $\mathbf{E}_2$ avoids name capture. A reduction is *imperative* (resp. *functional*) when it is derived using a communication along an imperative (resp. functional) name.

A process P_0 *diverges* (or *is divergent*) if there is an infinite sequence of processes $\{P_i\}_{i\geq 0}$, such that, for all $i \geq 0$, $P_i \longrightarrow P_{i+1}$. Then P *terminates* (or *is terminating*) if P is not divergent; similarly, a set of processes is terminating if all its elements are.

Subcalculi. $\pi_{\mathtt{def}}$ is the subcalculus with only functional names (i.e., without the productions in the second line of the grammar for processes: restriction, linear and replicated inputs), and $\pi_{\mathtt{imp}}$ is the purely imperative one (i.e., without the `def` construct).

Termination constraints and logical relation proofs for λ-calculi can be adapted to $\pi_{\mathtt{def}}$. In the simply-typed case, the only additional constraint is that definitions $\mathtt{def}\ f = P\ \mathtt{in}\ Q$ cannot be recursive (f is not used in the body P). We call $\pi^1_{\mathtt{def}}$ the language obtained in this way. It is rather easy to show that $\pi^1_{\mathtt{def}}$ is included in the languages studied in [YBH04, San06], which gives:

Theorem 1. *$\pi^1_{\mathtt{def}}$ is terminating.*

Various terminating sublanguages of $\pi_{\mathtt{imp}}$ have been defined in the literature. One of the simplest ones, in [DS06], ensures termination by a straightforward measure-decreasing argument. Roughly, an integer number, called *level*, is attached to each name; then one requires that in any input $a(x).P$, the level of each input-unguarded output in P is smaller than the level of a; in other words, the output at a consumed to activate P must be heavier than any output which is released. Thus the overall measure of a process (computed on the multiset of all its input-unguarded outputs) decreases at every reduction. The measure assignment is formalised as a type system to ensure invariance under reduction. We call $\pi^1_{\mathtt{imp}}$ this terminating language, derived by adapting to $\pi_{\mathtt{imp}}$ the simplest type system from [DS06] (this type system is hardwired into the system we present in Section 3).

3 Types for Termination in $\pi_{\mathtt{ST}}$

In this section we define a type system for termination that combines the constraints of the imperative $\pi^1_{\mathtt{imp}}$ with those of the functional $\pi^1_{\mathtt{def}}$. First note that a straightforward merge of the two languages can break termination. This is illustrated in

$$\mathtt{def}\ f = \overline{a}\ \mathtt{in}\ (!a.\overline{f} \mid \overline{a}) \ ,$$

where a recursion on the name a is fed via a recursion through the functional definition of f. The process is divergent, yet the constraints in $\pi^1_{\mathtt{def}}$ and $\pi^1_{\mathtt{imp}}$ are respected (the process is simply-typed, has no recursive definition, the imperative input $!a.\overline{f}$ is well-typed as the level of a can be greater than the level of f).

Termination is guaranteed if the measure constraint imposed on imperative names is extended to the functional ones, viewing a `def` as an input. This extension would however be too naive: it would indeed affect the class of functional processes accepted, which would be rather limited (not containing for instance the calculus $\pi^1_{\mathtt{def}}$ and the process images of the simply-typed λ-calculus).

In the solution we propose, we do impose measures onto the functional names, but the constraints on them are more relaxed. This way, $\pi^1_{\mathtt{def}}$ is captured. The drawback is that the measure alone does not anymore guarantee termination. However, it does if the purely functional sublanguage (in our case $\pi^1_{\mathtt{def}}$) is terminating.

$$(\textbf{Res})\ \frac{\vdash_\Gamma P : l}{\vdash_\Gamma (\boldsymbol{\nu} a)\, P : l} \qquad (\textbf{Par})\frac{\vdash_\Gamma P_1 : l_1 \qquad \vdash_\Gamma P_2 : l_2}{\vdash_\Gamma P_1 \mid P_2 : \max(l_1, l_2)}$$

$$(\textbf{Nil})\frac{}{\vdash_\Gamma \mathbf{0} : 0} \qquad (\textbf{Out})\ \frac{\vdash_\Gamma P : l \qquad \vdash_\Gamma c : \sharp^k_\bullet T \qquad \vdash_\Gamma w : T}{\vdash_\Gamma \overline{c}\langle w\rangle.P : \max(k, l)}$$

$$(\textbf{In})\ \frac{\begin{array}{c}\vdash_\Gamma P : l \qquad \vdash_\Gamma a : \sharp^k_{\mathtt{I}}\, T \\ \vdash_\Gamma x : T \qquad k > l\end{array}}{\vdash_\Gamma a(x).P : 0} \qquad (\textbf{Rep})\ \frac{\begin{array}{c}\vdash_\Gamma P : l \qquad \vdash_\Gamma a : \sharp^k_{\mathtt{I}}\, T \\ \vdash_\Gamma x : T \qquad k > l\end{array}}{\vdash_\Gamma\, !a(x).P : 0}$$

$$(\textbf{Def})\ \frac{\begin{array}{c}\vdash_\Gamma P_1 : l \qquad \vdash_\Gamma P_2 : l' \qquad \vdash_\Gamma f : \sharp^k_{\mathtt{F}}\, T \\ k \geq l \qquad f \notin \mathrm{fn}(P_1)\end{array}}{\vdash_\Gamma \mathtt{def}\ f = (x).P_1\ \mathtt{in}\ P_2 : l'}$$

Fig. 2. Typing Rules for $\pi^1_{\mathtt{ST}}$

The whole constraints are formalised as a refinement of the simply-typed system in which a level is attached to types. To ease the proofs, the system is presented *à la Church*: every name has a predefined type. Thus a typing Γ is a total function from names to types, with the proviso that every type is inhabited by an infinite number of names. We write $\Gamma(x) = T$ to mean that x has type T in Γ. Types are given by:

$$T \ ::= \ \sharp^k_{\mathtt{F}}\, T \ \big|\ \sharp^k_{\mathtt{I}}\, T \ \big|\ \mathtt{unit} \ .$$

Type $\sharp^k_{\mathtt{F}}\, T$ is assigned to functional names that carry values of type T; similarly for $\sharp^k_{\mathtt{I}}\, T$ and imperative names. In both cases, k is a natural number called the *level*. We write $\sharp^k_\bullet\, T$ when the functional/imperative tag is not important.

The typing judgement for processes is of the form $\vdash_\Gamma P : l$, where l is the *level* (or weight) *of* P. It is defined by the rules of Figure 2; on values, $\vdash_\Gamma v : T$ holds if either $\Gamma(v) = T$, or $v = \star$ and $T = \mathtt{unit}$.

We comment on the definition of the type system. Informally, the level of a process P indicates the maximum level of an input-unguarded output in P. Condition $k > l$ in rules (**In**) and (**Rep**) implements the measure constraint for $\pi^1_{\mathtt{imp}}$ discussed in Section 2. In rule (**Def**) the constraint is looser: the outputs released can be as heavy as the one being consumed (condition $k \geq l$). In other words, we just check that functional reductions do not cause violations of the stratification imposed on the imperative inputs (which would happen if a functional output underneath an imperative input a could cause the release of imperative outputs heavier than a, as in the example at the beginning of this section). In the same rule (**Def**), the constraint $f \notin \mathrm{fn}(P_1)$ is inherited from $\pi^1_{\mathtt{def}}$, and forbids recursive calls in functional definitions.

We call $\pi^1_{\mathtt{ST}}$ the set of well-typed processes. Our type system subsumes the existing ones, and satisfies the subject reduction property:

Lemma 2. *We have* $\pi^1_{\mathtt{def}} \subseteq \pi^1_{\mathtt{ST}}$ *and* $\pi^1_{\mathtt{imp}} \subseteq \pi^1_{\mathtt{ST}}$.

Proposition 3. *If $\vdash_\Gamma P : l$ and $P \longrightarrow P'$ then $\vdash_\Gamma P' : l'$ for some l'.*

The termination proof for $\pi^1_{\mathtt{imp}}$ uses the property that at every reduction the *multiset measure* of a process (the multiset of the levels of input-unguarded outputs in the process) decreases. In $\pi^1_{\mathtt{ST}}$, this only holds for imperative reductions: along functional reductions the overall weight may actually increase. It would probably be hard to establish termination of $\pi^1_{\mathtt{ST}}$ by relying only on measure arguments.

Inferring termination. In view of the results in [DHKS07], we believe that a type inference procedure running in polynomial time can be defined for the type system of Figure 2. This is possible as long as typability (and hence termination) can be inferred in polynomial time for the core calculus (here, $\pi^1_{\mathtt{def}}$). The situation is less clear if we work in an impure π-calculus without syntactical distinction (via the `def` construct) between imperative and functional names, replicated inputs being used both for functional definitions and imperative inputs.

4 Termination Proof

In the proof, we take a well-typed $\pi^1_{\mathtt{ST}}$ process, assume that it is divergent, and then derive a contradiction. Using a pruning function (Definition 4) and a related simulation result (Lemma 7), we transform the given divergence into one for a functional process in $\pi^1_{\mathtt{def}}$. This yields contradiction, as $\pi^1_{\mathtt{def}}$ is terminating (Theorem 1).

Thus the definition of pruning is central in our proof. Intuitively, pruning computes the *functional backbone* of a process P, by getting rid of imperative outputs and replacing imperative inputs (replicated or not) in P with the simplest possible functional term, namely $\mathbf{0}$. However, to establish simulation, we would need reduction to commute with pruning (possibly replacing reduction with a form of "semantic equality" after commutation). This however does not hold, at least using a simple definition of pruning: take for instance $P_0 \overset{\text{def}}{=} !a.\overline{f} \,|\, \overline{a} \longrightarrow P_1 \overset{\text{def}}{=} !a.\overline{f} \,|\, \overline{f}$; the pruning of P_0 would be $\mathbf{0}$ whereas that of P_1 would be $\overline{f}$, and the latter processes cannot be related in a natural way.

We therefore have to be more precise, and make pruning parametric on a level p that occurs infinitely often in the reductions of the given divergent computation (cf. Lemma 8 – the level of a reduction is the level of the name along which the reduction occurs). Further, we define p as the maximal level that occurs infinitely often in the divergent computation (Lemma 8). Thus, at the cost of possibly removing an initial segment of the computation, we can assume the absence of reductions at levels greater than p. Indeed the actual pruning computes the *functional backbone at level p* of a process: we remove all imperative constructs, *and* the functional constructs (`def` and outputs) acting on functional names whose level is smaller than p. Typing ensures us that, in doing so, no functional constructs at level p that participate in the infinite computation are

$$\mathsf{pr}^p_\Gamma(a(x).P) = \mathsf{pr}^p_\Gamma(!a(x).P) = \mathsf{pr}^p_\Gamma(\mathbf{0}) \overset{\text{def}}{=} \mathbf{0}$$

$$\mathsf{pr}^p_\Gamma(P_1 \mid P_2) \overset{\text{def}}{=} \mathsf{pr}^p_\Gamma(P_1) \mid \mathsf{pr}^p_\Gamma(P_2)$$

$$\mathsf{pr}^p_\Gamma((\boldsymbol{\nu} a)\, P) \overset{\text{def}}{=} (\boldsymbol{\nu} a)\, \mathsf{pr}^p_\Gamma(P) \qquad \mathsf{pr}^p_\Gamma(\overline{a}\langle v\rangle.P) \overset{\text{def}}{=} \mathsf{pr}^p_\Gamma(P)$$

$$\mathsf{pr}^p_\Gamma(\mathtt{def}\ f^n = (x).P_1\ \mathtt{in}\ P_2) \overset{\text{def}}{=}$$

$$\begin{cases} \mathtt{def}\ f = (x).\mathsf{pr}^p_\Gamma(P_1)\ \mathtt{in}\ \mathsf{pr}^p_\Gamma(P_2) & \text{if } n = p \\ \mathsf{pr}^p_\Gamma(P_2) & \text{otherwise} \end{cases}$$

$$\mathsf{pr}^p_\Gamma(\overline{f}^n\langle v\rangle.P) \overset{\text{def}}{=} \begin{cases} \overline{f}\langle v\rangle.\mathsf{pr}^p_\Gamma(P) & \text{if } n = p \\ \mathsf{pr}^p_\Gamma(P) & \text{otherwise} \end{cases}$$

Fig. 3. Pruning in $\pi^1_{\mathtt{ST}}$

removed. Therefore, the functional reductions at level p in the original divergent computation are preserved by the pruning, while the other kinds of reductions have no semantic consequence on pruning (in the sense that processes before and after the reduction are pruned onto semantically equal terms). We thus derive an infinite computation in $\pi^1_{\mathtt{def}}$, which is impossible as $\pi^1_{\mathtt{def}}$ is terminating.

We can remark in passing that in the example given above, P_0 is pruned to $\mathbf{0}$, and so is P_1: indeed, the level of a is less or equal than p (otherwise we would not examine a reduction of P_0), so that f's level is strictly smaller than p, which entails, by Definition 4 below, that the output on f disappears when pruning P_1: in this case, the prunings of the terms involved in the reduction are equal.

Pruning and its properties. The definition and properties of the pruning function are parametric upon a level p (which will be fixed in Lemma 8 below) and a typing Γ.

Definition 4 (Pruning). *The pruning of P w.r.t. p and Γ, written $\mathsf{pr}^p_\Gamma(P)$, is defined by induction on P as in Figure 3, where a (resp. f^n) is a name whose type in Γ is imperative (resp. functional with level n).*

We write $\mathrm{PR}(\Gamma)$ for the typing context of the simply-typed π-calculus obtained from Γ by removing all level information from the types, and $\vdash^\pi_\Gamma Q$ for the typing judgements in the simply-typed π-calculus. We rely on these notations to state the following lemma:

Lemma 5. *Suppose $\vdash_\Gamma P : l$. Then, for any p, $\vdash^\pi_{\mathrm{PR}(\Gamma)} \mathsf{pr}^p_\Gamma(P)$.*

We write $\simeq$ for *strong barbed congruence*, defined in the usual way [SW01]. $\simeq$ is needed in the statement of the two next lemmas — we essentially use that $\simeq$ preserves divergences and is closed by static contexts.

The following technical lemma establishes that in certain cases, the pruning of the continuation of an input is inactive, up to $\simeq$, and can thus be removed. This property is useful to derive the key simulation result involving pruning (Lemma 7).

Lemma 6 (Pruning – inactivity).

- *If* $\vdash_\Gamma$ `def` $f = (x).P_1$ `in` $P_2 : n$, *with* $\Gamma(f) = \sharp^n_{\mathsf{F}} T$ *and* $n < p$, *then for any* v *s.t.* $\Gamma(v) = T$, $\mathsf{pr}^p_\Gamma(P_1)\{v/x\} \simeq \mathbf{0}$.
- *If* $\vdash_\Gamma !a(x).P_1 : n$, *with* $\Gamma(a) = \sharp^n_{\mathsf{F}} T$ *and* $n \le p$, *then for any* v *s.t.* $\Gamma(v) = T$, $\mathsf{pr}^p_\Gamma(P_1)\{v/x\} \simeq \mathbf{0}$.
- *If* $\vdash_\Gamma a(x).P_1 : n$, *with* $\Gamma(a) = \sharp^n_{\mathsf{F}} T$ *and* $n \le p$, *then for any* v *s.t.* $\Gamma(v) = T$, $\mathsf{pr}^p_\Gamma(P_1)\{v/x\} \simeq \mathbf{0}$.

Note that these properties are not immediate: for instance, in $a(x).P$, the continuation P may contain top-level definitions at level p, that are not removed by pruning. We write $P \longrightarrow^n_{\mathrm{I}} P'$ (resp. $P \longrightarrow^n_{\mathrm{F}} P'$) if P has a reduction to P' in which the interacting name is imperative (resp. functional) and its level is n.

Lemma 7 (Simulation)
Suppose $\vdash_\Gamma P : l$.

1. *If* $P \longrightarrow^p_{\mathrm{F}} P'$, *then* $\mathsf{pr}^p_\Gamma(P) \longrightarrow \mathsf{pr}^p_\Gamma(P')$;
2. *If* $P \longrightarrow^n_{\mathrm{F}} P'$ *with* $n < p$, *then* $\mathsf{pr}^p_\Gamma(P) \simeq \mathsf{pr}^p_\Gamma(P')$;
3. *If* $P \longrightarrow^n_{\mathrm{I}} P'$ *with* $n \le p$, *then* $\mathsf{pr}^p_\Gamma(P) \simeq \mathsf{pr}^p_\Gamma(P')$.

A delicate aspect of the proof of Lemma 7 is that if, for instance, $!a(x).P$ is involved in a reduction, an instantiated copy of P is unleashed at top-level; we rely on Lemma 6 to get rid of it (modulo $\simeq$).

The impossibility of an infinite computation. We can now present the termination proof, which exploits pruning to derive a contradiction from the existence of an infinite computation starting from a typable process.

Lemma 8. *If* $\vdash_\Gamma P_0 : l$, *and if there is an infinite computation starting from* P_0, *given by* $\{P_i\}_{i\ge0}$, *then there exists a maximum level* p *such that there are infinitely many reductions at level* p *in the sequence. Moreover, there are infinitely many functional reductions at level* p.

Proof (sketch). *The first part holds because a process is typed using a finite number of levels.*

For the second part, we exploit the fact that the multiset of input-unguarded outputs at level p *in a process decreases strictly along imperative reductions at level* p, *and decreases or remains equal along (imperative or functional) reductions at levels strictly smaller than* p.

Theorem 9 (Soundness). *All processes in* π^1_{ST} *are terminating.*

Proof (sketch). *By absurd, let* $\{P_i\}_{i\ge0}$ *be an infinite computation starting from a* π^1_{ST} *process* P_0 *such that* $\vdash_\Gamma P_0 : l$ *for some* Γ *and* l. *By Proposition 3, all the* P_i*'s are well-typed. By Lemma 8 there is a maximal* p *s.t. for infinitely many* i, $P_i \longrightarrow^p_{\mathrm{F}} P_{i+1}$. *Moreover, still by Lemma 8, we can suppose w.l.o.g. that no reduction occurs at a level greater than* p.

Whenever $P_i \longrightarrow^p_{\mathrm{F}} P_{i+1}$, we obtain by Lemma 7 $\mathsf{pr}^p_\Gamma(P_i) \longrightarrow \mathsf{pr}^p_\Gamma(P_{i+1})$. Lemma 7 also tells us that when P_i reduces to P_{i+1} via a reduction that is not a functional reduction at level p, we have $\mathsf{pr}^p_\Gamma(P_i) \simeq \mathsf{pr}^p_\Gamma(P_{i+1})$. This allows us to construct an infinite computation in $\pi^1_{\mathtt{def}}$ (by Lemma 5), which is impossible (Theorem 1).

5 Parametrisation on the Core Calculus

In this and the following sections we discuss how the exposed method can be enhanced and enriched.

In Section 4, we assume a termination proof for (a subset of) the core functional $\pi_{\mathtt{def}}$ from which the impure terms are built, but we never look into the details of such proof. As a consequence, the method can be made parametric. Let F be the chosen terminating functional core, and L the impure π-calculus language that we want to prove terminating. There are three requirements on L:

1. processes in L are well-typed in the type system of Figure 2, but without the constraint $f \notin \mathrm{fn}(P_1)$ of rule (**Def**) (that represented a specific constraint coming from $\pi^1_{\mathtt{def}}$);
2. L is reduction closed (i.e., $P \in L$ and $P \longrightarrow P'$ imply $P' \in L$);
3. the pruning of a process in L (as given by Definition 4) returns a process in F.

Theorem 10. *If F and L satisfy (1-3) above, then L is terminating.*

It may be possible to formulate constraints for condition (3) as typing constraints to be added to the type system used in condition (1), as we actually did in Section 3, where F is $\pi^1_{\mathtt{def}}$ and L is $\pi^1_{\mathtt{ST}}$.

An example of a functional core that could be chosen in place of $\pi^1_{\mathtt{def}}$ is $\pi^{\mathrm{pr}}_{\mathtt{def}}$, the π-calculus image of the primitive-recursive functions. Its termination is enforced by a control of recursion; the constraint on a construct $\mathtt{def}\ f = (x).P\ \mathtt{in}\ Q$ is that for any input-unguarded output $\overline{c}\langle v\rangle$ in P: (i) either the level of f is higher than that of c, or (ii) the two names have the same level and carry a first-order argument (e.g., a natural number) that decreases its value ($v < x$).

Even when the termination of the functional core can be proved using measures, as is actually the case for $\pi^{\mathrm{pr}}_{\mathtt{def}}$, the parametrisation on this core offered by the method could be useful, because the measures employed in the proofs for the functional core and for the whole language can be different; see example $Sys_{2'}$ discussed at the end of Section 6.

Iterating the method. Our method could also be applied to a purely functional calculus in which names are partitioned into two sets, and the measure technique guarantees termination only for the reductions along names that belong to one of the sets. An example is $\pi^{1+\mathrm{pr}}_{\mathtt{def}}$, the calculus combining $\pi^1_{\mathtt{def}}$ and $\pi^{\mathrm{pr}}_{\mathtt{def}}$; thus, in any construct $\mathtt{def}\ f = (x).P_1\ \mathtt{in}\ P_2$, either f is not used in P_1, or all outputs in

P_1 at the same level as f emit values provably smaller than x. (We think that a natural encoding of System T [GLT88] in the π-calculus would not be typable in $\pi^{1+\mathrm{pr}}_{\mathtt{def}}$.)

The method can be iteratively applied. For instance, having used it to establish termination for the functional calculus $\pi^{1+\mathrm{pr}}_{\mathtt{def}}$, as suggested above, we can then take $\pi^{1+\mathrm{pr}}_{\mathtt{def}}$ as the functional core of a new application of the method. This iteration could for instance add imperative names. In [DHS10a], we use an iteration of the method to prove the termination of an example that combines the two examples presented in Section 6.

6 Examples

This section shows examples of the kind of systems whose termination can be proved with the method exposed in the earlier sections. These systems make use of both imperative and functional names. The systems cannot be handled by previous measure-based techniques, which are too weak on functional names, or by logical relation techniques, which are too weak on imperative names. Our method offers the capability of combining techniques for both kinds of names, which is essential.

The examples use polyadicity, and first-order values such as integers and related constructs (including arithmetic constructs, and if-then-else statements). These extensions are straightforward to accommodate in the theory presented earlier. A longer example is given in [DHS10a].

An encryption server. In this example, several clients c_i are organised into a chain. Clients are willing to communicate, however direct communications between them are considered unsafe. Instead, each client must use a secured channel s to contact a server that is in charge of encrypting and sending the information to the desired client. Hence the messages travel through the c_i's in order to be finally emitted on d. A client c_i, receiving a message, has to perform some local imperative atomic computations. For readability, we condense this part into the acquire and release actions of a local lock named $lock_i$.

Several messages can travel along the chain concurrently, and may overtake each other; the example is stated with n initial messages (they are all sent to c_1 but could have been directed to other clients).

$$\begin{array}{l} Sys_1 \overset{\text{def}}{=} (\boldsymbol{\nu} lock_1, .., lock_k) \\ \qquad \Big(\; lock_1 \mid .. \mid lock_k \mid \\ \qquad\quad \mathtt{def}\ s = (c, x).\overline{c}\langle \mathbf{enc}[c, x]\rangle\ \mathtt{in} \\ \qquad\qquad \mathtt{def}\ c_1 = C_1\ \mathtt{in}\ \ \ldots\ \mathtt{def}\ c_{k-1} = C_{k-1}\ \mathtt{in}\ \mathtt{def}\ c_k = C_k\ \mathtt{in} \\ \qquad\qquad\quad (\overline{s}\langle c_1, \mathrm{msg}_1\rangle \mid \ldots \mid \overline{s}\langle c_1, \mathrm{msg}_n\rangle) \Big) \end{array}$$

where C_i $(1 \le i < k)$ and C_k are:

$$\begin{array}{l} C_i \overset{\text{def}}{=} (y_i).\overline{lock_i}.(lock_i \mid \overline{s}\langle c_{i+1}, \mathbf{dec}_i[y_i]\rangle) \\ C_k \overset{\text{def}}{=} (y_k).\overline{lock_k}.(lock_k \mid \overline{d}\langle \mathbf{dec}_k[y_k]\rangle) \end{array}$$

and **enc**, **dec** are operators for encryption and decryption, with equalities $\mathbf{dec}_i[\mathbf{enc}[c_i, m]] = m$ for all i.

The way the c_i's are organised (here a chain, but any well-founded structure could have been used) ensures that the interactions taking place among the server and the clients do not lead to divergent behaviours: this is guaranteed by showing that the system is terminating.

In the typing all the c_i's must have the same type, because they all appear as the first component of messages on s. We can type-check Sys_1 using the system of Section 3 (where the core functional language is $\pi^1_{\mathtt{def}}$); if $\mathtt{b}$ is the type for the encrypted/decrypted data, then we assign $\sharp^1_{\mathtt{F}}\,\mathtt{b}$ for the c_i's, $\sharp^1_{\mathtt{F}}\,(\sharp^1_{\mathtt{F}}\,\mathtt{b} \times \mathtt{b})$ for s, and $\sharp^1_{\mathtt{I}}\,\mathtt{unit}$ for the $lock_i$'s.

The loose assignment of levels to functional names (the possibility $k = l$ in rule (**Def**) of Section 3) is essential for the typing: an output on c_i can follow an input on s on the server's side, and conversely on the clients' side: c_i and s must therefore have the same level.

A movie-on-demand server. In this second example, the server s is a movie database, and handles requests to watch a given movie (in streaming) a given number of times. By sending the triple $\langle 15, r', \mathbf{tintin}\rangle$ on s, the client pays for the possibility to watch the movie **tintin** 15 times; r' (r in the server's definition) is the return channel, carrying the URL (h) where the movie will be made available once.

Two loops are running along the execution of the protocol. On the server's side, a recursive call is generated with $n - 1$, after consumption of one access right. On the client's side, process $!c.r'(z).(\overline{c} \mid z)$ keeps interrogating the server: the client tries to watch the movie as many times as possible.

$$
\begin{array}{rcl}
Sys_2 & \stackrel{\mathrm{def}}{=} & \mathtt{def}\ s = (n,r,f).\\
 & & \quad\big(\ \ \mathtt{if}\ f = \mathbf{tintin}\ \mathtt{then}\ (\nu h)\ (\overline{r}\langle h\rangle.\overline{h})\\
 & & \quad\ \ \mid \mathtt{if}\ f = \mathbf{asterix}\ \mathtt{then}\ \ldots\\
 & & \quad\ \ \ldots\\
 & & \quad\ \ \mid \mathtt{if}\ n > 0\ \mathtt{then}\ \overline{s}\langle n-1,r,f\rangle\ \big)\\
 & & \mathtt{in}\ \ (\boldsymbol{\nu} r')\ \big(\ \overline{s}\langle 15, r', \mathbf{tintin}\rangle \mid (\boldsymbol{\nu} c)\,(\overline{c} \mid !c.r'(z).(\overline{c}\mid z))\ \big)
\end{array}
$$

Here again, we rely on typing to guarantee that this system, where a server is able to call itself recursively, does not exhibit divergences.

Sys_2 uses both functional names (e.g., s) and imperative names (e.g., r). Its termination is proved by appealing to the primitive recursive language $\pi^{\mathrm{pr}}_{\mathtt{def}}$ as core functional calculus. In the typing, channel c is given level 1, and s, r level 2 (this allows us to type-check the recursive output on c).

To understand the typing of functional names, it may also be useful to consider the variant $Sys_{2'}$ in which the following definition of a server s' is inserted between the definition of s and the body $(\boldsymbol{\nu} r')\ldots$:

$$\mathtt{def}\ s' = (n,r,f).\overline{s}\langle n,r,f\rangle\ \mathtt{in}\ ..$$

Here, s' models an old address where the server, now at s, used to run. Some of the clients might still use s' instead of s, and s' hosts a forwarder that redirects their requests to s.

We can type $Sys_{2'}$ thanks to the looser level constraints on functional names which allow s and s' to have the same type; the functional core is still $\pi^{\mathrm{pr}}_{\mathtt{def}}$. Note that a termination proof of the core calculus $\pi^{\mathrm{pr}}_{\mathtt{def}}$ by a simple measure argument is not in contradiction with the observation that similar measures are too weak to type $Sys_{2'}$: the levels used in the typing of $Sys_{2'}$ need not be the same as those used in the termination proof of its functional core (the pruning of $Sys_{2'}$). Indeed in this functional core s' can be given a level higher than that of s (which is possible because the imperative clients have been pruned, hence, *after the pruning*, s and s' need not have the same type).

7 Refinements

Non-replicated imperative inputs. In the type system of Figure 2, non-replicated inputs introduce the same constraint $k > l$ as replicated inputs (rules (**In**) and (**Rep**)). This can be annoying, as the inputs that are dangerous for termination are the replicated ones. Simply removing the constraint $k > l$ from rule (**In**) would however lead to an unsafe system. For instance, we could type the divergent process

$$\mathtt{def}\ f = (x).(\overline{a}\langle x\rangle \mid \overline{x})\ \mathtt{in}\ \mathtt{def}\ g = a(y).\overline{f}\langle y\rangle\ \mathtt{in}\ \overline{f}\langle g\rangle\ .$$

This example shows some of the subtle interferences between functional and imperative constructs that may cause divergences.

We can improve rule (**In**) so to impose the constraint $k > l$ only when the value communicated in the input is functional. Rule (**In**) is replaced by the following rule, where "T functional" holds if T is of the form $\sharp^{k'}_{\mathtt{F}}\ T'$, for some k', T':

$$(\mathbf{In}')\,\frac{\vdash_\Gamma P : l \qquad \vdash_\Gamma a : \sharp^{k}_{\mathtt{I}}\, T \qquad \vdash_\Gamma x : T \qquad \text{if } T \text{ functional then } k > l \text{ and } l' = 0\text{, otherwise } l' = l}{\vdash_\Gamma a(x).P : l'}$$

Corresponding modifications have to be made in the definition of pruning and related results. The rule could be refined even further, by being more selective on the occurrences of x in P when x is functional. An example of this is discussed in [DHS10a], to be able to treat name c in the process Sys_2, from Section 6, as functional. (It is also possible to avoid communications of functional names along imperative names, by simple program transformations whereby communication of a functional name is replaced by communication of an imperative name that acts as a forwarder towards the functional name.)

A benefit of these refinements is to be able to accept processes $a(x).P$ where a is imperative and appears in input-unguarded outputs of P. This typically arises in the modelling of mutable variables in the asynchronous π-calculus (references can also be modelled as services that accept read and write requests; in this case the above refinements of the input rule are not needed).

Another refinement of the measure-based analysis, in which we handle *sequences* of input prefixes, along the lines of [DS06], is presented in [DHS10a].

Polymorphism. The method can be extended to handle polymorphism [Tur96]. A level on each type should however be maintained, and type abstraction therefore cannot hide levels. A type variable X is hence annotated with a level k, and it should only be instantiated with types whose level is less than or equal to k (assuming that first-order types like integers have level 0).

8 Conclusions

We have described an analysis to guarantee termination of impure languages. Our technique exploits pruning in order to lift a termination proof based on logical relations (for a functional language) to a termination proof for a richer language.

Despite recent progresses in impure simply-typed λ-calculi, adapting the realisability/logical relation technique to non-trivial impure concurrent languages such as π-calculus remains hard. All our attempts failed, intuitively because, on the one hand, typing naturally arises in π-calculus on names and less naturally on terms; on the other hand, even adopting forms of behavioural types for π-calculus terms, the main operator for composing processes is parallel composition, and this operator does not isolate computation (in contrast with λ-calculus application, which forces two terms to interact with each other). Using restricted forms of parallel composition in π-calculus often forbids useful imperative patterns.

The extensions of realisability recently studied by Boudol [Bou07] and Amadio [Ama09] for impure λ-calculi can also handle extra features such as threads. However the threads in these works do not interfere with termination. Most importantly, the underlying language must be the λ-calculus (the technique relies on the standard realisability, and the crucial type construct in proofs is the functional type). It does not seem possible to transport these proofs outside the λ-calculus, e.g., in the π-calculus.

In another extension of the realisability technique, Blanqui [Bla04] is able to lift a termination proof based on realisability for simply-typed λ-calculus into a termination proof for a λ-calculus with higher-order pattern-matching operators, provided a well-founded order exists on the constructors used for pattern-matching. Again, as refinement of the realisability approach for a λ-calculus, the method proposed, and the details of the proof, are entirely different from ours.

In the paper, we have proposed a method to treat the problem in which a standard measure-based technique for imperative languages is made parametric with respect to a terminating functional core language; the termination of the core language is established separately. The method can be further enhanced by refining the initial measure technique. The core language parameter of the method is functional. We do not know how to relax the property, needed in the definition of pruning.

Acknowledgements. We wish to thank G. Boudol for fruitful discussions in the initial stages of this work. Support from the french ANR projects "CHoCo" and ANR-08-BLANC-0211-01 "Complice", and by the European Project "HATS" (contract number 231620) is also acknowledged.

References

[Ama09] Amadio, R.: On Stratified Regions. In: Hu, Z. (ed.) APLAS 2009. LNCS, vol. 5904, pp. 210–225. Springer, Heidelberg (2009)

[Bla04] Blanqui, F.: A type-based termination criterion for dependently-typed higher-order rewrite systems. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 24–39. Springer, Heidelberg (2004)

[Bou07] Boudol, G.: Fair Cooperative Multithreading. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 272–286. Springer, Heidelberg (2007)

[DHKS07] Demangeon, R., Hirschkoff, D., Kobayashi, N., Sangiorgi, D.: On the Complexity of Termination Inference for Processes. In: Barthe, G., Fournet, C. (eds.) TGC 2007 and FODO 2008. LNCS, vol. 4912, pp. 140–155. Springer, Heidelberg (2008)

[DHS10a] Demangeon, R., Hirschkoff, D., Sangiorgi, D.: Appendix to this paper, proofs (2010), http://www.cs.unibo.it/~sangio/

[DHS10b] Demangeon, R., Hirschkoff, D., Sangiorgi, D.: A Method for Termination in a λ-calculus with References (2010), http://www.cs.unibo.it/~sangio/

[DS06] Deng, Y., Sangiorgi, D.: Ensuring Termination by Typability. Information and Computation 204(7), 1045–1082 (2006)

[GLT88] Girard, J.-Y., Lafont, Y., Taylor, P.: Proofs and Types. Cambridge Tracts in Theoretical Computer Science, vol. 7. Cambridge University Press, Cambridge (1988)

[PT00] Pierce, B.C., Turner, D.N.: Pict: a programming language based on the pi-calculus. In: Proof, Language, and Interaction, pp. 455–494. The MIT Press, Cambridge (2000)

[San06] Sangiorgi, D.: Termination of Processes. Math. Structures in Comput. Sci. 16(1), 1–39 (2006)

[SW01] Sangiorgi, D., Walker, D.: The π-calculus: a Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)

[Tur96] Turner, N.D.: The polymorphic pi-calculus: Theory and Implementation. PhD thesis, Department of Computer Science, University of Edinburgh (1996)

[YBH04] Yoshida, N., Berger, M., Honda, K.: Strong Normalisation in the Pi-Calculus. Information and Computation 191(2), 145–202 (2004)

Buffered Communication Analysis in Distributed Multiparty Sessions[*]

Pierre-Malo Deniélou and Nobuko Yoshida

Department of Computing, Imperial College London

Abstract. Many communication-centred systems today rely on asynchronous messaging among distributed peers to make efficient use of parallel execution and resource access. With such asynchrony, the communication buffers can happen to grow inconsiderately over time. This paper proposes a static verification methodology based on multiparty session types which can efficiently compute the upper bounds on buffer sizes. Our analysis relies on a uniform causality audit of the entire collaboration pattern — an examination that is not always possible from each end-point type. We extend this method to design algorithms that allocate communication channels in order to optimise the memory requirements of session executions. From these analyses, we propose two refinements methods which respect buffer bounds: a *global protocol refinement* that automatically inserts confirmation messages to guarantee stipulated buffer sizes and a *local protocol refinement* to optimise asynchronous messaging without buffer overflow. Finally our work is applied to overcome a buffer overflow problem of the multi-buffering algorithm.

1 Introduction

Session types for buffer bound analysis. The expensive cost of synchronous communications has led programmers to rely on asynchronous messaging for efficient network interactions. The downside is that non-blocking IO requires buffers that can grow inconsiderately over time, bringing systems to stop. The analysis and debugging of this phenomenon is mainly done by a tedious monitoring of the communicated messages of the whole distributed system. This paper shows that, when a global interaction pattern is explicitly specified as a *multiparty session* [11,1,15,22], types can provide an effective way to statically verify buffer usage and communication optimisations, automatically guaranteeing safe and deadlock-free runs.

Session types, first introduced in [20,10], can specify communication protocols by describing the sequences and types of read, write and choices on a given channel. For example, type $T_0 = !\langle \mathsf{nat} \rangle ; !\langle \mathsf{string} \rangle ; ?\langle \mathsf{real} \rangle ; \mathsf{end}$, in the original binary session type syntax, expresses that a nat-value and string-value will be sent in that order, then that a real-value is expected as an input, and finally that the protocol ends.

We can use session types to calculate the upper bounds of the buffer sizes of asynchronous channels (message passing is non-blocking and order-preserving using FIFO buffers). For example, from type T_0, we can compute that the maximum number of messages that might be stored in a communication buffer is two, while a different type

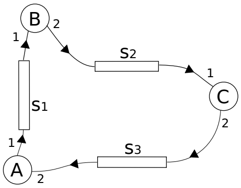
[*] The work is partially supported by EPSRC EP/F003757/01 and EPSRC G015635/01.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 343–357, 2010.

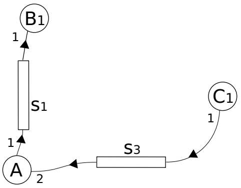
$T_1 = !\langle \mathsf{nat} \rangle; ?\langle \mathsf{real} \rangle; !\langle \mathsf{string} \rangle; \mathsf{end}$ guarantees a maximum size of one, since the dual process engaged with T_1 is forced to consume a nat-value before sending the next real-message. This use of session types is informally observed in [7] and formally studied in [8] for binary session types. However, the binary case does not yield a direct extension to multiparty interactions as explained below.

Buffer bounds analysis in multiparty sessions. We start by illustrating the difficulties of such an analysis on a simple three party interaction (Example (a) below), where $s!\langle V \rangle$ is an output of V to s, $s?(x);P$ an input at s, and $\mu X.P$ a recursive agent:

Example (a)
(A) Alice=$\mu X.s_1!\langle 1 \rangle; s_3?(x); X$
(B) Bob =$\mu X.s_1?(x); s_2!\langle Orange \rangle; X$
(C) Carol=$\mu X.s_2?(x); s_3!\langle 2.4 \rangle; X$
Example (b)
(B$_1$) Bob$_1$ =$\mu X.s_1?(x); X$
(C$_1$) Carol$_1$=$\mu X.s_3!\langle 2.4 \rangle; X$



Example (a)



Example (b)

We assume the three buffers of s_1, s_2 and s_3 are initially empty and that values are pushed and read one by one. Assuming session types ensure that accesses to buffers do not create any race condition at any moment of the infinite protocol execution, none of the channels s_1, s_2, s_3 need to buffer more that one value at any given time.

However, if we change Bob and Carol to Bob$_1$ and Carol$_1$ as Example (b) above, while they still interact correctly, the buffers of s_1 and s_3 need an unbounded size because of the *lack of synchronisation* between Bob$_1$ and Carol$_1$. The main difficulty of the communication buffer analysis is that, unlike in binary session types, each endpoint type itself does not provide enough information: for example, Alice's local type $T_a = \mu \mathbf{x}.s_1!\langle \mathsf{nat} \rangle; s_3?\langle \mathsf{real} \rangle; \mathbf{x}$ (repeatedly sends a nat-value to s_1 and receives a real-value from s_3) is *the same* in both Examples (a) and (b), while the needed buffer size for s_1 and s_3 are different (1 in (a) and ∞ in (b)) due to the change in the other parties' behaviours. Our first question is: *can we statically and efficiently determine the upper size of buffers in multiparty interactions?* In our case, we take advantage of the existence of a global session type [11,1,15,22] for the analysis:

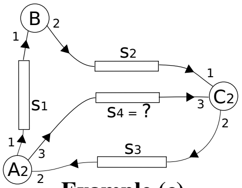
$$G = \mu \mathbf{x}.\text{Alice} \to \text{Bob}: s_1 \langle \mathsf{nat} \rangle; \text{Bob} \to \text{Carol}: s_2 \langle \mathsf{string} \rangle; \text{Carol} \to \text{Alice}: s_3 \langle \mathsf{real} \rangle; \mathbf{x}$$

The above type represents the global interaction between Alice-Bob-Carol in (a) where Alice → Bob: $s_1 \langle \mathsf{nat} \rangle$; means that Alice sends a nat-value to Bob through buffer s_1. To analyse buffer usage, we consider sessions as graphs and track *causal chains* for each channel: alternated message production and consumption mark the execution points at which buffers are emptied. This can be observed in Example (a). On the other hand, the global type of Alice-Bob$_1$-Carol$_1$ in (b) lacks the second Bob → Carol: no message forces Carol to wait for Bob's reception before sending the next message. In that case, each buffer may accumulate an unbounded number of messages.

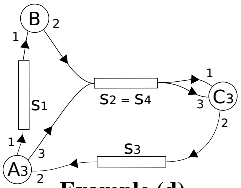
Channel allocation. Our next problem is about resource allocation. *Given a global scenario, can we assign the minimum number of resources (channels) without conflict* so that, for instance, we can efficiently open a minimal number of sockets for a given network interaction? Assume Alice and Carol in (a) wish to communicate one more message after completing three communications, where the new communication happens on a fresh channel s_4 (Example (c) below). Can we reuse either s_1, s_2 or s_3 for this new communication? Reusing s_1 creates a writing conflict (the order between Alice's first and third messages would be unspecified) and reusing s_3 would create a reading conflict (Carol could read her own message).

Example (c)
(A_2) $\text{Alice}_2 = \mu X.s_1!\langle 1\rangle;s_3?(x);s_4!\langle x{+}1\rangle;X$
(B) $\text{Bob} = \mu X.s_1?(x);s_2!\langle \mathit{Orange}\rangle;X$
(C_2) $\text{Carol}_2 = \mu X.s_2?(x);s_3!\langle 2.4\rangle;s_4?(y);X$

Example (d)
(A_3) $\text{Alice}_3 = \mu X.s_1!\langle 1\rangle;s_3?(x);s_2!\langle x{+}1\rangle;X$
(C_3) $\text{Carol}_3 = \mu X.s_2?(x);s_3!\langle 2.4\rangle;s_2?(y);X$



Example (c)



Example (d)

The only safe way to reuse a channel in Example (c) is to merge s_2 and s_4 as in Example (d), in which case communications on other channels prevent any conflict.

Global refinement for multiparty sessions. The third issue is how to fix a buffer overflow problem by "global refinement", i.e. alteration of the original global protocol to satisfy given buffer sizes. Here, our simple approach is the insertion of a minimal number of *confirmation messages* to enforce synchronisation. In network or business protocols, they can be implemented as a system level signal. Consider the interaction (b) among Alice-Bob$_1$-Carol$_1$ where each buffer requires an unbounded size. If we wish to enforce a buffer size of at most 2, we can build a new global type where one confirmation message from Bob to Carol is inserted in any second iteration as:

$$\begin{aligned} G' \;=\; \mu\mathbf{x}.\;& \text{Alice} \to \text{Bob}: s_1\,\langle \mathsf{nat}\rangle;\text{Carol} \to \text{Alice}: s_3\,\langle \mathsf{real}\rangle; \\ & \text{Alice} \to \text{Bob}: s_1\,\langle \mathsf{nat}\rangle;\text{Bob} \to \text{Carol}: s_2\,\langle \mathsf{string}\rangle;\text{Carol} \to \text{Alice}: s_3\,\langle \mathsf{real}\rangle;\mathbf{x} \end{aligned}$$

The revised processes following G' are given as:

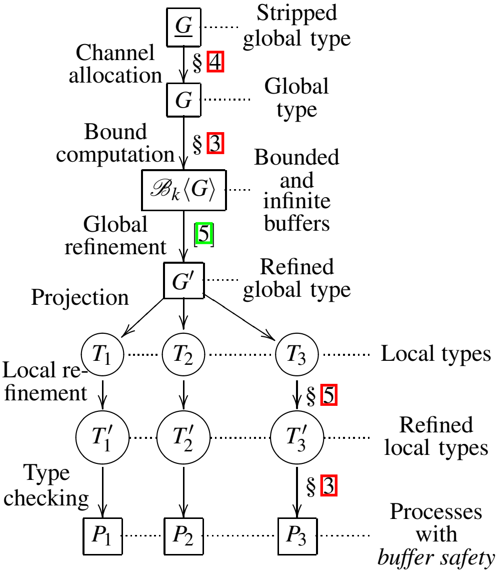
$$\text{Bob}_4 = \mu X.s_1?(x);s_1?(x);s_2!\langle \mathit{Signal}\rangle;X \qquad \text{Carol}_4 = \mu X.s_3!\langle 2.4\rangle;s_2?(x).s_3!\langle 2.4\rangle;X$$

Local refinement for multiparty messaging optimisations. The last issue is about flexible local refinement (optimisations) based on [15,14]. Assume that, in Example (a), Bob wishes to always start the asynchronous transmission of the string *Orange* to the buffer s_3 without waiting for the delivery of the first nat-value from Alice on s_1.

$$\text{Bob}_5 \;=\; \mu X.s_2!\langle \mathit{Orange}\rangle;s_1?(x);X \qquad (1.1)$$

Due to Bob's unilateral implementation change, all three minimal buffer sizes go up from 1 to 2. Moreover, suppose Bob repeatedly applies the same optimisation on his next n messages, as in $s_2!\langle \mathit{Orange}\rangle;s_2!\langle \mathit{Orange}\rangle;..;s_2!\langle \mathit{Orange}\rangle;$Bob. While the result is communication-safe (no mismatch of the communication with Carol), all three minimal buffer sizes go up from 1 to n. *How can we perform local optimisation without altering buffer sizes in a multiparty session*?

Contributions are summarised in the figure below. To the best of our knowledge, our work is the first which guarantees safe buffered multiparty communications for the π-calculus with communication-safety, progress and flexible refinements. The key contribution is a general causal analysis over graphs constructed from multiparty session types (§ 3). Due to the space limitation, we omit the global refinement. A full version which includes the global refinement, omitted definitions, examples and proofs, and a prototype for computing the upper bound on the buffer size are available [5].



1. The overall analysis starts from global types that have no channel annotation. We attribute channels based on memory requirements (§ 4).
2. From global types, our bound analysis computes the buffer bounds of finite channels and finds the infinite ones (§ 3)
3. The global refinement method then introduces additional messages to prevent any unboundedness (long version [5]).
4. Once the global type has been projected to local types, local refinement can optimise the distributed execution of the participants' processes (§ 5).
5. The running optimised processes can then be typed and enjoy communication, buffer and type safety and progress properties (§ 3).
6. We apply our work to the multibuffering algorithm (§ 5.1).

2 Asynchronous Multiparty Sessions

Syntax. We start from the π-calculus for multiparty sessions from [11] with unbounded and bounded buffers. Base sets and the grammars are given below.

$P ::= \overline{a}[2..n](\tilde{s}^{\tilde{m}}).P \mid a[p](\tilde{s}).P$	request, accept	$a,b,x,y,..$	shared names
$\mid s!\langle\tilde{e}\rangle;P \mid s?(\tilde{x});P$	send, receive	$s,t,..$	session channels
$\mid s!\langle\!\langle\tilde{s}\rangle\!\rangle;P \mid s?(\!(\tilde{s})\!);P$	session send, receive	$l,l',..$	labels
$\mid s\triangleleft l;P \mid s\triangleright\{l_i\colon P_i\}_{i\in I}$	selection, branch	$X,Y,..$	process variables
$\mid \text{if } e \text{ then } P \text{ else } Q$	conditional	$m,n,..$	buffer size (integers or ∞)
$\mid \mathbf{0} \mid (\nu a)P \mid (\nu\tilde{s})P$	inact, hiding	$e ::= v \mid e \text{ and } e' \cdots$	expressions
$\mid P \mid Q \mid \mu X.P \mid X$	par, recursion	$v ::= a \mid \text{true} \mid \text{false} \cdots$	values
$\mid s^n : \tilde{h}$	message buffer	$h ::= l \mid \tilde{v} \mid \tilde{t}$	message values

$\overline{a}[2..n](\tilde{s}^{\tilde{m}}).P$ initiates, through a shared name a, a new session s_i with buffer size m_i ($1 \leq n \leq \infty$) with other participants, each of the form $a[p](\tilde{s}).Q$ with $1 \leq p \leq n-1$. The s_i in vector $\tilde{s}$ is a session channel (bounded by buffer size m_i) used in the session. We call p, q,... (natural numbers) the *participants* of a session. Session communications (which take place inside an established session) are performed by the sending and receiving of a value; the session sending and reception (where the former delegates to the latter the capability to participate in a session by passing a channel associated with the session which is called *delegation*); and by selection and branching (the former chooses one of the branches offered by the latter). $s^n : \tilde{h}$ is a *message buffer of size n* representing ordered messages in transit $\tilde{h}$ with destination s. This may be considered as a network pipe in a TCP-like transport with fixed bandwidth. The rest of the syntax is standard from [11]. We often omit n from $s^n : \tilde{h}$, $\mathbf{0}$, and unimportant arguments e.g. $s!\langle\rangle$ and $s?();P$. An *initial* process does not contain any runtime syntax (buffers and session hiding).

Reductions. A selection of reduction rules is given below.

$$\overline{a}[2..\mathtt{n}]\,(\tilde{s}^{\tilde{n}}).P_1 \mid a[2]\,(\tilde{s}).P_2 \mid \cdots \mid a[\mathtt{n}]\,(\tilde{s}).P_{\mathtt{n}} \;\to\; (\nu\,\tilde{s})(P_1 \mid P_2 \mid \ldots \mid P_{\mathtt{n}} \mid s_1^{n_1}:\emptyset \mid \ldots \mid s_m^{n_m}:\emptyset)$$

$$\begin{array}{ll} s!\langle\tilde{e}\rangle;P \mid s^n:\tilde{h} \;\to\; P \mid s^n:\tilde{h}\cdot\tilde{v}\;\; (n \gneq |\tilde{h}|,\, e_i \downarrow v_i) & s?(\tilde{x});P \mid s^n:\tilde{v}\cdot\tilde{h} \to P[\tilde{v}/\tilde{x}] \mid s^n:\tilde{h} \\ s!\langle\!\langle\tilde{t}\rangle\!\rangle;P \mid s^n:\tilde{h} \;\to\; P \mid s^n:\tilde{h}\cdot\tilde{t}\;\; (n \gneq |\tilde{h}|) & s?(\!(\tilde{t})\!);P \mid s^n:\tilde{t}\cdot\tilde{h} \to P \mid s^n:\tilde{h} \\ s \lhd l;P \mid s^n:\tilde{h} \;\to\; P \mid s^n:\tilde{h}\cdot l\;\; (n \gneq |\tilde{h}|) & s \rhd \{l_i\colon P_i\}_{i\in I} \mid s^n:l_j\cdot\tilde{h} \to P_j \mid s^n:\tilde{h} \qquad (j \in I) \end{array}$$

The first rule describes the initiation of a new session among n participants that synchronise over the shared name a. After the initiation, they will share m fresh private session channels s_i and the associated m empty buffers of size n_m ($\emptyset$ denotes an empty queue). The output rules for values, sessions and selection respectively enqueue values, sessions and labels if the buffer is not full. $e_i \downarrow v_i$ denotes the evaluation of e_i to v_i. We define $|\emptyset| = 0$ and $|\tilde{h}\cdot h| = |\tilde{h}| + 1$. The size $n = \infty$ corresponds to the original asynchronous unbounded buffered semantics [11]. The input rules perform the complementary operations. Processes are considered modulo a structural equivalence $\equiv$, whose definition is standard (e.g. $\mu X.P \equiv P[\mu X.P/X]$) [11].

3 Bound Analysis in Multiparty Sessions

This section presents an analysis of causal chains and buffer sizes and introduces the typing system for the *buffer safety* property (Corollary 3.9).

3.1 Global Types and Dependencies

Global types. A *global type*, written by $G, G', ..$, describes the whole conversation scenario of a multiparty session as a type signature. Our starting syntax is from [11].

$$\begin{array}{rcll} G, G' & ::= & \mathtt{p} \to \mathtt{p}'\colon k\,\langle U\rangle;G' & \text{values} \\ & \mid & \mathtt{p} \to \mathtt{p}'\colon k\,\{l_j\colon G_j\}_{j\in J} & \text{branching} \\ & \mid & \mu\mathbf{x}.G \mid \mathbf{x} \mid \mathsf{end} & \text{recursion, end} \end{array} \qquad \begin{array}{rcll} U, U' & ::= & \tilde{S} \mid T@\mathtt{p} & \text{sorts, session} \\ S, S' & ::= & \mathsf{bool} \mid \mathsf{nat} \mid G & \text{base, shared} \end{array}$$

Type $\mathtt{p} \to \mathtt{p}'\colon k\,\langle U\rangle;G'$ says that participant p sends a message of type U on the channel k (represented as a natural number) so that participant $\mathtt{p}'$ can receive it. The session continues with the interactions described in G'. The *value types* U, U' are either a vector of sorts or a *located type* $T@\mathtt{p}$, representing a local type T assigned to participant p. Located types are used for delegation and defined in § 3.3. *Sorts* S, S' are either base types or global types for shared names. Type $\mathtt{p} \to \mathtt{p}'\colon k\,\{l_j\colon G_j\}_{j\in J}$ says that participant p can invoke one of the l_i labels on channel k (for participant $\mathtt{p}'$ to read) and that interactions described in G_j follow. We require $\mathtt{p} \neq \mathtt{p}'$ to prevent self-sent messages. Type $\mu\mathbf{x}.G$ is for recursive protocols, assuming the type variables $(\mathbf{x}, \mathbf{x}', \ldots)$ are guarded in the standard way, i.e. they only occur under values or branchings. We assume G in value types is closed, i.e. without free type variables. Type end represents session termination (often omitted). $k \in G$ means k appears in G. The functions $\mathit{chans}(G)$ and $\mathit{prins}(G)$ respectively give the number of channels and participants of G.

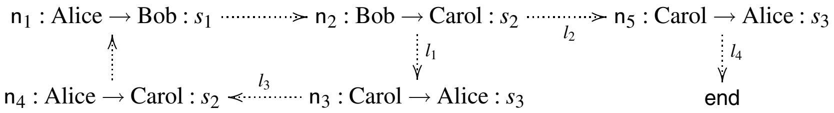
Sessions as graphs. Global types can be seen (isomorphically) as *session graphs*, that we define in the following way. First, we annotate in G each syntax occurrence of subterms of the form $\mathsf{p} \to \mathsf{p}' \colon k\,\langle U\rangle; G'$ or $\mathsf{p} \to \mathsf{p}' \colon k\,\{l_j \colon G_j\}_{j\in J}$ with a node name $(\mathsf{n}_1, \mathsf{n}_2, \ldots)$. Then, we inductively define a function $\mathtt{node}_G$ that gives a node n_k (or the special node end) for each of the syntactic subterm of G as follows:

$$\begin{array}{rlrl}
\mathtt{node}_G(\mathsf{end}) = \mathsf{end} & & \mathtt{node}_G(\mathsf{n}_i : \mathsf{p} \to \mathsf{p}' \colon k\,\langle U\rangle; G') & = \mathsf{n}_i \\
\mathtt{node}_G(\mu\mathbf{x}.G') = \mathtt{node}_G(G') & & \mathtt{node}_G(\mathsf{n}_j : \mathsf{p} \to \mathsf{p}' \colon k\,\{l_j \colon G_j\}_{j\in J}) & = \mathsf{n}_j \\
\mathtt{node}_G(\mathbf{x}) = \mathtt{node}_G(\mu\mathbf{x}.G') & \textit{(if the binder of } \mathbf{x} \textit{ is } \mu\mathbf{x}.G' \in G\textit{)} & &
\end{array}$$

We define G as a session graph in the following way: for each subterm of G of the form $\mathsf{n} : \mathsf{p} \to \mathsf{p}' \colon k\,\langle U\rangle; G'$, we have an edge from n to $\mathtt{node}_G(G')$, and for each subterm of G of the form $\mathsf{n}' : \mathsf{p} \to \mathsf{p}' \colon k\,\{l_j \colon G_j\}_{j\in J}$, we have edges from n' to each of the $\mathtt{node}_G(G_j)$ for $j \in J$. We also define the functions $\mathtt{pfx}(\mathsf{n}_i)$ and $\mathtt{ch}(\mathsf{n}_i)$ that respectively give the prefix ($\mathsf{p} \to \mathsf{p}' : k$) and channel ($k$) that correspond to n_i. For a global type G, $\mathtt{node}_G(G)$ distinguishes the *initial* node. $\mathit{size}(G)$ denotes the number of edges of G.

Example 3.1 (Session graph). Our running example extends Example (a) from § 1 with branching. Below, we give the global type followed by its graph representation, with the edges as the dotted arrows (labels are for information). n_1 is the initial node.

$$\begin{array}{l}
\mu\mathbf{x}.\text{Alice} \to \text{Bob} \colon s_1\,\langle \mathsf{nat}\rangle; \text{Bob} \to \text{Carol} : s_2 \\
\qquad\qquad\qquad\qquad \left\{\begin{array}{l} l_1 : \text{Carol} \to \text{Alice} : s_3\{l_3 : \text{Alice} \to \text{Carol} \colon s_2\,\langle \mathsf{string}\rangle; \mathbf{x}\} \\ l_2 : \text{Carol} \to \text{Alice} : s_3\{l_4 : \mathsf{end}\} \end{array}\right\}
\end{array}$$



The recursion call yields a cycle in the graph, while branching gives the edges l_1 and l_2.

The edges of a given session graph G define a successor relation between nodes, written $\mathsf{n} \prec \mathsf{n}'$ (omitting G). Paths in this session graph are referred to by the sequence of nodes they pass through: a path $\mathsf{n}_0 \prec \ldots \prec \mathsf{n}_n$ can be written more concisely $\mathsf{n}_0 \ldots \mathsf{n}_n$ or $\tilde{\mathsf{n}}$ when there is no ambiguity. We say that a path $\mathsf{n}_0 \ldots \mathsf{n}_n$ *has suffix* $\mathsf{n}_i \ldots \mathsf{n}_n$ for $0 < i < n$. The empty path is ε. The transitive closure of $\prec$ is $\prec\!\!\prec$.

IO-chains. We detect causality chains in a given G by the relation $\prec_{\mathtt{IO}}$, defined below:

$$\mathsf{n}_1 \prec_{\mathtt{IO}} \mathsf{n}_2 \text{ if } \mathsf{n}_1 \prec\!\!\prec \mathsf{n}_2 \text{ and } \mathtt{pfx}(\mathsf{n}_1) = \mathsf{p}_1 \to \mathsf{p} : k_1 \text{ and } \mathtt{pfx}(\mathsf{n}_2) = \mathsf{p} \to \mathsf{p}_2 : k_2 \text{ with } k_1 \neq k_2$$

The relation $\prec_{\mathtt{IO}}$ asserts the order between a reception by a principal and the next message it sends. An *input-output dependency (IO-dependency)* from n_1 to n_n is a chain $\mathsf{n}_1 \prec_{\mathtt{IO}} \cdots \prec_{\mathtt{IO}} \mathsf{n}_n$ $(n \geq 1)$.

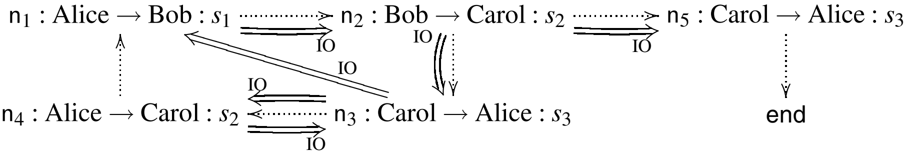
3.2 Algorithms for Buffer Size Analysis

Unbounded buffers. In some sessions, messages (sent asynchronously) can accumulate without moderation in a buffer. A simple test can predict which channels require an unbounded buffer. We use the fact that IO-dependencies characterise the necessity for a channel buffer to be emptied before proceeding. Infinite channels are the ones where such a dependency is missing.

Definition 3.2 (infinite and finite). A channel k is said to be *finite* if, for every node $\mathsf{n} \in G$ and for every cycle $\tilde{\mathsf{n}}$ for $\prec$ such that $\mathtt{ch}(\mathsf{n}) = k$ and $\mathsf{n} \in \tilde{\mathsf{n}}$, there exists a cycle for $\prec_{\mathtt{IO}}$ that starts from n and only involves nodes from $\tilde{\mathsf{n}}$. The other channels are *infinite*.

Correspondingly, buffers are said to be *bounded* or *unbounded*. Given G, checking for the infinity of k in G can be computed in a time bounded by $O(size(G)^3)$. The proof relies on the fact that establishing all IO-dependencies of a given session has $O(size(G)^3)$ time-complexity (assuming the number of participants as a constant).

Example 3.3 (Session graph and infinite channels). We illustrate on our running example the previous notions. We add to the picture the IO-dependencies (with $\Rightarrow$).


n_1 : Alice → Bob : s_1
n_2 : Bob → Carol : s_2
n_5 : Carol → Alice : s_3
IO
IO
IO
IO
IO
IO
n_4 : Alice → Carol : s_2
n_3 : Carol → Alice : s_3
end

Since each node of the main cycle $\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3\mathsf{n}_4$ is part of the IO-cycles $\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3$ or $\mathsf{n}_3\mathsf{n}_4$, there are no infinite channels in this session.

Counting finite buffer size. To compute the bounds on buffer sizes, we first need to define a property on paths that characterises when a buffer has to be emptied.

Definition 3.4 (reset). If $\tilde{\mathsf{n}} = \mathsf{n}_0 \ldots \mathsf{n}_n\mathsf{n}$ is a path in G, the property Reset($\tilde{\mathsf{n}}$) holds if there exist $0 \le i_0 < \ldots < i_j \le n$ $(j \ge 1)$ such that $\mathsf{n}_{i_0} \prec_{\mathtt{IO}} \ldots \prec_{\mathtt{IO}} \mathsf{n}_{i_j} \prec_{\mathtt{IO}} \mathsf{n}$ and $\mathtt{ch}(\mathsf{n}_{i_0}) = \mathtt{ch}(\mathsf{n})$. One practical instance of the nodes $\{\mathsf{n}_{i_0}, \ldots, \mathsf{n}_{i_j}, \mathsf{n}\}$ is called the reset nodes of $\tilde{\mathsf{n}}$.

The paths that satisfy the reset property are the ones for which there exists a reception guard to the last node.

Now that we know which buffers are infinite and have characterised the resetting paths that control buffer growth, we can describe our algorithm to count the buffer size required by finite channels. For each channel k of a global session type G, we define a function $\mathscr{B}_k\langle G\rangle$ that will compute the bound on the buffer size of channel k. The key step is to reset the counter when we recognise the appropriate IO-dependencies.

Definition 3.5 (bound computation). Given a session graph G, for each channel k, we compute the bound as $\mathscr{B}_k\langle G\rangle = \mathscr{B}_k\langle 0, \emptyset, \varepsilon, \mathsf{n}_0\rangle$ for n_0 the initial node of G.

$$\mathscr{B}_k\langle m, \mathscr{P}, \tilde{\mathsf{n}}, \mathsf{n}\rangle = \begin{cases} 0 & \text{if } \mathsf{n} = \mathsf{end} \text{ or } \tilde{\mathsf{n}} \in \mathscr{P} \\ \max_{\mathsf{n} \prec \mathsf{n}'} \mathscr{B}_k\langle m, \{\tilde{\mathsf{n}}\} \cup \mathscr{P}, \tilde{\mathsf{n}}\mathsf{n}, \mathsf{n}'\rangle & \text{if } \mathtt{ch}(\mathsf{n}) = k', k \neq k' \\ \max_{\mathsf{n} \prec \mathsf{n}'} \mathscr{B}_k\langle 1, \{\tilde{\mathsf{n}}\} \cup \mathscr{P}, \mathsf{n}, \mathsf{n}'\rangle & \text{if } \mathtt{ch}(\mathsf{n}) = k, \text{Reset}(\tilde{\mathsf{n}}\mathsf{n}) \\ \max(m+1, \max_{\mathsf{n} \prec \mathsf{n}'} \mathscr{B}_k\langle m+1, \{\tilde{\mathsf{n}}\} \cup \mathscr{P}, \tilde{\mathsf{n}}\mathsf{n}, \mathsf{n}'\rangle) & \text{if } \mathtt{ch}(\mathsf{n}) = k, \neg\text{Reset}(\tilde{\mathsf{n}}\mathsf{n}) \end{cases}$$

The algorithm explores all the paths of the session graph until they grow to satisfy the reset property. Since we examine only finite channels, the length of such paths is limited and the algorithm terminates. The bound on the buffer size of a channel is the maximum buffer size required over these paths. For each path, the algorithm acts recursively on the edges and maintains a counter (m in $\mathscr{B}_k\langle m, \mathscr{P}, \tilde{\mathsf{n}}, \mathsf{n}\rangle$) that records the current size of the buffer. If the current prefix does not involve the channel k, the size of the buffer is unchanged and the computation continues to the next nodes. If the current prefix uses the channel k, there are two cases: (a) the reset property holds for the current path, in which case the buffer has been emptied prior to the current message; or (b) the reset property does not hold and the buffer needs to be able to keep one more value. When there are no further node, or when the path currently examined has already been considered (i.e. is in $\mathscr{P}$), the algorithm stops.

Given a global type G, the upper bound of channel k in G can be computed in polynomial time. Note that the computation can be done for all channels at once.

Example 3.6 (buffer bound analysis). We illustrate the algorithm on our running session example, where we compute the bound for channel s_2 (we omit $\mathscr{P}$ for readability):

$\mathscr{B}_{s_2}\langle 0,\varepsilon,\mathsf{n}_1\rangle$	max	explanation
$= \mathscr{B}_{s_2}\langle 0,\mathsf{n}_1,\mathsf{n}_2\rangle$	0	$s_1 \neq s_2$
$= \max(\mathscr{B}_{s_2}\langle 1,\mathsf{n}_1\mathsf{n}_2,\mathsf{n}_3\rangle,\mathscr{B}_{s_2}\langle 1,\mathsf{n}_1\mathsf{n}_2,\mathsf{n}_5\rangle)$	1	$\neg\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2),\neg\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_3)$
$= \max(\mathscr{B}_{s_2}\langle 1,\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3,\mathsf{n}_4\rangle,\mathscr{B}_{s_2}\langle 1,\mathsf{n}_1\mathsf{n}_2\mathsf{n}_5,\mathsf{end}\rangle)$	1	$s_3 \neq s_2$
$= \max(\mathscr{B}_{s_2}\langle 1,\mathsf{n}_4,\mathsf{n}_1\rangle,0)$	1	$\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3\mathsf{n}_4)$
$= \mathscr{B}_{s_2}\langle 1,\mathsf{n}_4\mathsf{n}_1,\mathsf{n}_2\rangle$	1	$s_1 \neq s_2$
$= \max(\mathscr{B}_{s_2}\langle 2,\mathsf{n}_4\mathsf{n}_1\mathsf{n}_2,\mathsf{n}_3\rangle,\mathscr{B}_{s_2}\langle 2,\mathsf{n}_4\mathsf{n}_1\mathsf{n}_2,\mathsf{n}_5\rangle)$	2	$\neg\mathrm{Reset}(\mathsf{n}_4\mathsf{n}_1\mathsf{n}_2),\neg\mathrm{Reset}(\mathsf{n}_4\mathsf{n}_1\mathsf{n}_3)$
$= \max(\mathscr{B}_{s_2}\langle 2,\mathsf{n}_4\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3,\mathsf{n}_4\rangle,\mathscr{B}_{s_2}\langle 2,\mathsf{n}_4\mathsf{n}_1\mathsf{n}_2\mathsf{n}_5,\mathsf{end}\rangle)$	2	$s_3 \neq s_2$
$= \max(\mathscr{B}_{s_2}\langle 1,\mathsf{n}_4,\mathsf{n}_1\rangle,0)$	**2**	$\mathrm{Reset}(\mathsf{n}_4\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3\mathsf{n}_4)$

The algorithm starts with n_1, the root of G. Since n_1 uses buffer s_1 (different from s_2), we continue with the successor n_2. It uses s_2 and, since the accumulated path $\mathsf{n}_1\mathsf{n}_2$ does not satisfy the reset property, the buffer requirement of s_2 needs to be increased to 1. The next nodes, n_3 and n_5, do not use the channel s_2. Since n_4 uses s_2 and $\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3\mathsf{n}_4)$ holds (there is $\mathsf{n}_2 \prec_{\mathtt{IO}} \mathsf{n}_3 \prec_{\mathtt{IO}} \mathsf{n}_4$), the buffer has to be emptied before n_4: we thus reinitialise the buffer requirement to 1 and the path to just n_4. On the other branch, we reach end and stop the computation. The next prefix of n_4, n_1, does not use s_2, but it successor n_2 does. We thus check the reset property on the path $\mathsf{n}_4\mathsf{n}_1\mathsf{n}_2$, but it does not hold. The buffer requirement is thus increased to 2. As previously, n_3 and n_5 do not use the channel s_2 and the accumulated path (in the main branch) becomes $\mathsf{n}_4\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3$. The next prefix, n_4, uses s_2 and $\mathrm{Reset}(\mathsf{n}_4\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3\mathsf{n}_4)$ holds: thus we initialise the buffer requirement back to 1 and the path to just n_4. However, we just explored such a situation earlier in the computation and thus stop. The maximum buffer size encountered for s_2 is then 2. Such a computation for s_1 and s_3 gives a buffer size of 1.

3.3 Subject Reduction and Buffer Safety

Once global type G is agreed upon by all parties, a local type T_i from each party's viewpoint is generated as a projection of G, and implemented as a process P_i. If all the resulting local processes $P_1,..,P_n$ can be type-checked against $T_1,..,T_n$, they are automatically guaranteed to interact properly, without communication mismatch (communication safety) nor getting stuck inside a session (progress) [11]. Here we additionally ensure the absence of buffer-overflow based on the buffer bound analysis of G.

Local types. Local session types type-abstract sessions from each end-point's view.

$$T \ ::= \ k!\langle U\rangle;T \ \mid \ k?\langle U\rangle;T \ \mid \ k\oplus\{l_i\colon T_i\}_{i\in I} \ \mid \ k\&\{l_i\colon T_i\}_{i\in I} \ \mid \ \mu\mathbf{x}.T \ \mid \ \mathbf{x} \ \mid \ \mathsf{end}$$

Type $k!\langle U\rangle$ expresses the sending to k of a value of type U. Type $k?\langle U\rangle$ is its dual. Type $k\oplus\{l_i\colon T_i\}_{i\in I}$ represents the transmission to k of a label l_i chosen in the set $\{l_i \mid i\in I\}$, followed by the communications described by T_i. Type $k\&\{l_i\colon T_i\}_{i\in I}$ is its dual. The remaining type constructors are standard. We say a type is *guarded* if it is neither a recursive type nor a type variable. The relation between global and local types is formalised by *projection*, written $G\upharpoonright \mathtt{p}$ (called *projection of* G *onto* $\mathtt{p}$) and defined in [11,22]. For example, $(\mathtt{p}\to\mathtt{p}'\colon k\langle U\rangle;G')\upharpoonright\mathtt{p} = k!\langle U\rangle;(G'\upharpoonright\mathtt{p})$, $(\mathtt{p}\to\mathtt{p}'\colon k\langle U\rangle;G')\upharpoonright\mathtt{p}' = k?\langle U\rangle;(G'\upharpoonright\mathtt{p}')$ and $(\mathtt{p}\to\mathtt{p}'\colon k\langle U\rangle;G')\upharpoonright\mathtt{q} = (G'\upharpoonright\mathtt{q})$. We take an *equi-recursive* view, not distinguishing between $\mu\mathbf{x}.T$ and its unfolding $T[\mu\mathbf{x}.T/\mathbf{x}]$.

Linearity. To avoid race conditions and conflicts between typed processes, we build on the definition of linearity from [11]. The relations $\prec_{\mathtt{II}}$ and $\prec_{\mathtt{OO}}$ are defined by:

$$\mathsf{n}_1 \prec_{\mathtt{II}} \mathsf{n}_2 \text{ if } \mathsf{n}_1 \prec\!\!\prec \mathsf{n}_2 \text{ and } \mathtt{pfx}(\mathsf{n}_1) = \mathtt{p}_1 \to \mathtt{p}\colon k_1 \text{ and } \mathtt{pfx}(\mathsf{n}_2) = \mathtt{p}_2 \to \mathtt{p}\colon k_2 \text{ s.t. } \mathtt{p}_1 \neq \mathtt{p}_2 \Leftrightarrow k_1 \neq k_2$$

$$\mathsf{n}_1 \prec_{\mathtt{OO}} \mathsf{n}_2 \text{ if } \mathsf{n}_1 \prec\!\!\prec \mathsf{n}_2 \text{ and } \mathtt{pfx}(\mathsf{n}_1) = \mathtt{p} \to \mathtt{p}_1\colon k_1 \text{ and } \mathtt{pfx}(\mathsf{n}_2) = \mathtt{p} \to \mathtt{p}_2\colon k_2 \text{ s.t. } \mathtt{p}_1 \neq \mathtt{p}_2 \Rightarrow k_1 \neq k_2$$

The three relations $\prec_{\mathtt{IO}}$, $\prec_{\mathtt{II}}$ and $\prec_{\mathtt{OO}}$ are used to characterise the authorised sequences of actions. An *input dependency (I-dependency) from* n_1 *to* n_2 is a chain $\mathsf{n}_1 \prec_{\phi_1} \cdots \prec_{\phi_n} \mathsf{n}_2$ $(n \geq 1)$ such that $\phi_i = \mathsf{IO}$ for $1 \leq i \leq n-1$ and $\phi_n = \mathsf{II}$. An *output dependency (O-dependency) from* n_1 *to* n_2 is a chain $\mathsf{n}_1 \prec_{\phi_1} \cdots \prec_{\phi_n} \mathsf{n}_2$ $(n \geq 1)$ such that $\phi_i \in \{\mathsf{OO},\mathsf{IO}\}$. These dependency relations are respectively written $\prec\!\!\prec_{\mathtt{II}}$ and $\prec\!\!\prec_{\mathtt{OO}}$. G is *linear* (written $\mathrm{Lin}(G)$) if, whenever two nodes $\mathsf{n}_1 \prec\!\!\prec \mathsf{n}_2$ use the same channel k, the dependencies $\mathsf{n}_1 \prec\!\!\prec_{\mathtt{II}} \mathsf{n}_2$ and $\mathsf{n}_1 \prec\!\!\prec_{\mathtt{OO}} \mathsf{n}_2$ hold. If G carries other global types, we inductively demand the same. Examples can be found in [11,5]. We call linear global types whose projections are defined *coherent*. Hereafter we only consider coherent types.

Typing initial processes. The type judgements for initial processes are of the form $\Gamma \vdash P \triangleright \Delta$ which means: "under the environment Γ, process P has typing Δ". Environments are defined by: $\Gamma ::= \emptyset \mid \Gamma, u:S \mid \Gamma, X:\Delta$ and $\Delta ::= \emptyset \mid \Delta, \tilde{s}^{\tilde{m}} : \{T@\mathtt{p}\}_{\mathtt{p}\in I}$. A *sorting* $(\Gamma,\Gamma',..)$ is a finite map from names to sorts and from process variables to sequences of sorts and types. *Typing* $(\Delta,\Delta',..)$ records linear usage of session channels. In multiparty sessions, it assigns a family of located types to a vector of session channels. In addition, we annotate each session channel s_k with its buffer bound m_k.

Among the typing rules, the rule for session initiation uses the buffer size $\mathscr{B}_{s_i}\langle G\rangle$ calculated from G.

$$\frac{\Gamma \vdash a\colon G \quad \Gamma \vdash P \triangleright \Delta, \tilde{s}^{\tilde{m}}\colon (G\upharpoonright \mathtt{1})@\mathtt{1} \;\; |\tilde{s}| = \mathit{chans}(G) \;\; \mathscr{B}_k\langle G\rangle = m_k}{\Gamma \vdash \overline{a}_{[2..\mathtt{n}]}(\tilde{s}^{\tilde{m}}).P \triangleright \Delta}$$

The type for $\tilde{s}$ is the *first* projection of the declared global type for a in Γ. The end-point type $(G\upharpoonright\mathtt{p})@\mathtt{p}$ means that the participant $\mathtt{p}$ has $G\upharpoonright\mathtt{p}$, which is the projection of G onto $\mathtt{p}$, as its end-point type. The condition $|\tilde{s}| = \mathit{chans}(G)$ means the number of session channels meets those in G. The condition $\mathscr{B}_k\langle G\rangle = m_k$ ensures that the size of the buffer m_i for each s_k does not exceed the size calculated from G. Similarly for accept. Other rules for initial processes are identical with [11]. Note that since $\mathscr{B}_k\langle G\rangle$ is decidable, type-checking for processes with type annotations is decidable [11,22].

The rest of the typing system for programs and one for runtime are similar with those in [11] ([5]). Judgements for runtime are there extended to $\Gamma \vdash_\Sigma P \triangleright \Delta$ with Σ a set of session channels associated to the current queue.

For the subject reduction, we need to keep track of the correspondence between the session environment and the buffer sizes. We use the reduction over session typing, $\Delta \xrightarrow{k} \Delta'$, that is generated by rules between types such as $k!\langle U\rangle;T\,@\mathtt{p}, k?\langle U\rangle;T'\,@\mathtt{q} \xrightarrow{k} T\,@\mathtt{p}, T'\,@\mathtt{q}$. The key lemma about the correspondence between buffer size and reduction follows. We set $[\![G]\!]$ to be the family $\{(G \upharpoonright \mathtt{p})@\mathtt{p} \mid \mathtt{p} \in G\}$. Regarding each type in $[\![G]\!]$ as the corresponding regular tree, we can define $\prec$, $\prec_{\mathtt{II}}, \prec_{\mathtt{IO}}$ and $\prec_{\mathtt{OO}}$ among its prefixes precisely as we have done for G.

Lemma 3.7. *If $\Delta(\tilde{s}) = [\![G]\!]$ and $\Delta \xrightarrow{s_k} \Delta'$, then $[\![G]\!](\xrightarrow{k})^*[\![G']\!]$ with $\Delta'(\tilde{s}) = [\![G']\!]$ and $\mathscr{B}_k\langle G\rangle \geq \mathscr{B}_k\langle G'\rangle$.*

When $\Gamma \vdash_\Sigma P \triangleright \Delta$, we say that $(\Gamma, \Sigma, P, \Delta)$ is *fully coherent* for session $\tilde{s}$ if there exist $P_1, \ldots, P_k, \Sigma', \Delta'$ such that $\Gamma \vdash_{\Sigma \uplus \Sigma'} P \mid P_1 \mid \ldots \mid P_k \triangleright \Delta, \Delta'$ and $\Delta, \Delta' = \Delta'', \tilde{s}^{\tilde{n}} : \{T_\mathtt{p}@\mathtt{p}\}_{\mathtt{p} \in I}$ with $[\![G]\!] = \{T_\mathtt{p}@\mathtt{p}\}_{\mathtt{p} \in I}$, G coherent and $\mathscr{B}_i\langle G\rangle \leq n_i$ $(1 \leq i \leq k)$.

Theorem 3.8 (Subject Reduction). *$\Gamma \vdash_\Sigma P \triangleright \Delta$ and $P \longrightarrow Q$ with $(\Gamma, \Sigma, P, \Delta)$ fully coherent imply $\Gamma \vdash_\Sigma Q \triangleright \Delta'$ for some Δ', s_k such that $\Delta = \Delta'$ or $\Delta(\xrightarrow{s_k})^*\Delta'$ and $\mathscr{B}_k\langle G\rangle \geq \mathscr{B}_k\langle G'\rangle$ with $\Delta(\tilde{s}) = [\![G]\!]$, $\Delta'(\tilde{s}) = [\![G']\!]$ and $(\Gamma, \Sigma, Q, \Delta')$ fully coherent.*

The proof relies on Lemma 3.7 and the fact that session reduction does not affect the causal dependencies within global types, so that buffer sizes can only decrease.

To state our buffer safety result, we define the *buffer overflow error* as follows:

$$\begin{array}{lcl} n \leq |\tilde{h}| & \Rightarrow & s!\langle \tilde{e}\rangle;P \mid s^n : \tilde{h} \rightarrow \mathsf{Err},\ s!\langle\!\langle \tilde{t}\rangle\!\rangle;P \mid s^n : \tilde{h} \rightarrow \mathsf{Err},\ s \lhd l;P \mid s^n : \tilde{h} \rightarrow \mathsf{Err} \\ P \rightarrow \mathsf{Err} & \Rightarrow & P \mid Q \rightarrow \mathsf{Err},\ (\nu\ a)P \rightarrow \mathsf{Err},\ (\nu\ \tilde{s})P \rightarrow \mathsf{Err},\ P \equiv Q \rightarrow \mathsf{Err} \end{array}$$

Corollary 3.9 (Buffer Safety) *If $\Gamma \vdash_\Sigma P \triangleright \Delta$, then for all P' s.t. $P \longrightarrow^* P'$, $P' \not\rightarrow \mathsf{Err}$.*

4 Channel Attribution

This section describes algorithms that attribute channels to the communications of a given global type without channels, called *stripped global types* ($\underline{G}, \underline{G'}, \ldots$) defined as:

$$\underline{G} ::= \ \ldots \mid \mathtt{p} \rightarrow \mathtt{p}'\langle U\rangle;\underline{G'} \mid \mathtt{p} \rightarrow \mathtt{p}'\{l_j : \underline{G_j}\}_{j \in J} \qquad \text{values, branching}$$

Our algorithms transform $\underline{G}$ into regular type G by adding channel annotations. We define the *channel allocation* of a global type G to be the value of the function $\mathtt{ch}$.

Singleton allocation. The simplest channel allocation attributes a different channel to each communication occurring in the global type syntax tree. Formally, the singleton allocation is such that: $\forall \mathsf{n}, \mathsf{n}' \in G,\ \mathtt{ch}(\mathsf{n}) = \mathtt{ch}(\mathsf{n}') \iff \mathsf{n} = \mathsf{n}'$. Singleton allocations enjoy the following good properties.

Lemma 4.1. *For any global type G with singleton allocation, (1) G satisfies the linearity property; (2) for the finite channels k of G, $\mathscr{B}_k\langle G\rangle = 1$; (3) for the finite channels k of G, $\sum_k \mathscr{B}_k\langle G\rangle \leq size(G)$.*

Channel equalities. As well as values of the ch function, allocations can be seen as partitions of the set of nodes $\{\mathsf{n}\}_{\mathsf{n}\in G}$. We then define partition refinement through the notion of *channel equality*, i.e. the union of two partitions to produce a new allocation.

Definition 4.2 (channel equality). A *channel equality* is the substitution of two channels k and k' in a global type G by a single fresh channel k'' while keeping G linear.

As we take the singleton allocation as a base, we can describe channel allocations by sets E of channel equalities, the empty set corresponding to the singleton allocation. We write G_E the global type G with channel equalities E.

In the rest of this section, we always start from the singleton allocation and proceed by channel equality. We notably rely on the fact that the result of the equality of two finite channels is finite. Formally, if $\mathscr{B}_k\langle G\rangle = \infty$ then $\forall E, \mathscr{B}_k\langle G_E\rangle = \infty$.

Note that the total number of possible channel allocations is finite and corresponds to the number of partitions of a given finite set. The exact count (if we do not take into account the linearity property) is given by a Bell number [19] which is exponential in the size of the global type. Given the finite number of possible allocations, we know that there exists an algorithm to find allocations satisfying any decidable property. Notably, one can reach any given memory requirement (number of channels, buffer sizes).

Principal allocation. The most widely used allocation method attributes two communication channels (one in each direction) for each pair of participants. The session types in [8,1] follow this allocation. Formally, the principal allocation is such that: $\forall \mathsf{n}, \mathsf{n}' \in G$ s.t. $\mathtt{pfx}(\mathsf{n}) = \mathsf{p} \to \mathsf{q} : k$ and $\mathtt{pfx}(\mathsf{n}') = \mathsf{p}' \to \mathsf{q}' : k', (k = k' \iff \mathsf{p} = \mathsf{p}' \wedge \mathsf{q} = \mathsf{q}')$.

Lemma 4.3. *For any global type G with principal allocation, (1) G satisfies the linearity property; (2) chans*$(G) \leq n \times (n-1)$ *where* $n = $ *prins*(G).

Greedy allocations. We now define a family of efficient algorithms, that give good allocation results in practice.

Definition 4.4 (Greedy allocation algorithm). *Given a global type with singleton allocation G, of initial node* n_0*, and a successor function* succ *over the nodes, the function* $\mathscr{I}_{\emptyset}^{\emptyset}(\mathsf{n}_0)$ *is defined by:*

$$\begin{array}{l} \mathscr{I}_E^{\mathrm{K}}(\mathsf{n}) = \mathscr{I}_{E'}^{\mathrm{K}'}(\mathsf{n}') \text{ where } \left\{\begin{array}{l} \mathtt{succ}(\mathsf{n}) = \mathsf{n}' \\ \mathtt{ch}(\mathsf{n}) = k \\ \mathrm{K}' = \mathrm{K} \cup \{k\} \end{array}\right. \wedge E' = \begin{cases} E \cup \{k = k'\} & \textit{if } \exists k' \in \mathrm{K}, \mathrm{Lin}(G_{E\cup\{k=k'\}}) \\ E & \textit{otherwise} \end{cases} \\ \mathscr{I}_E^{\mathrm{K}}(\mathsf{end}) = E \end{array}$$

This algorithm is parameterised by the successor function over nodes (that can be given e.g. by a depth-first graph search) and by the choice between the possible channels $k' \in \mathrm{K}$ for equality. The greedy algorithm has the advantage of not backtracking and thus enjoys a polynomial complexity (if the choice procedures are polynomial) in the size of the graph. In particular, we define two efficient heuristics based on the generic greedy algorithm. In the greedy algorithm, we implement K by either:

1. (Early) a queue, so that we choose for channel equality the oldest channel $k' \in \mathrm{K}$.
2. (Late) a list, so that we choose for channel equality the latest channel $k' \in \mathrm{K}$.

The early and late allocations are not optimal in terms of total memory requirements (computed by $\sum_k \mathscr{B}_k\langle G\rangle$ when all channels are finite) but give good results in practice while being polynomial.

Example 4.5 (comparison of the allocations). We apply the allocation algorithms on a three-party stripped global type. The results are given in the adjacent table in term of number of allocated channels and total memory requirement. The greedy algorithms give the best results on this example, with the early greedy algorithm allocating less channels than the late greedy algorithm.

	Singleton	Principal	Early G.	Late G.
$\mathsf{n}_0 : \mathtt{A} \to \mathtt{B};$	k_0	k_0	k_0	k_0
$\mathsf{n}_1 : \mathtt{B} \to \mathtt{A};$	k_1	k_1	k_1	k_1
$\mathsf{n}_2 : \mathtt{A} \to \mathtt{B};$	k_2	k_0	k_0	k_0
$\mathsf{n}_3 : \mathtt{A} \to \mathtt{B};$	k_3	k_0	k_0	k_1
$\mathsf{n}_4 : \mathtt{A} \to \mathtt{C};$	k_4	k_2	k_2	k_2
$\mathsf{n}_5 : \mathtt{C} \to \mathtt{B};$	k_5	k_3	k_1	k_3
$\mathsf{n}_6 : \mathtt{B} \to \mathtt{C};$	k_6	k_4	k_2	k_1
$\mathsf{n}_7 : \mathtt{B} \to \mathtt{C}$	k_7	k_4	k_2	k_2
Nb channels	8	5	3	4
Memory Req.	8	7	5	5

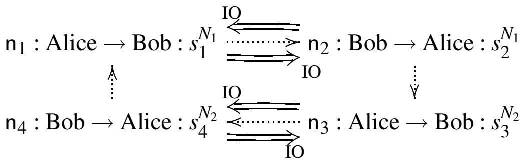
5 Local Refinement: Messaging Optimisations

One of the significant practical concerns in systems with messaging is to optimise interactions through more asynchronous data processing to increase parallelism. Our recent work [15,14] developed a new form of subtyping, the *asynchronous subtyping*, that characterises the compatibility between classes of type-safe permutations of actions, in order to send messages before receiving. This subtyping allows, however, not only Bob$_5$ in (1.1) in § 1 but also $\mu X.s_2!\langle Orange\rangle;X$ as a refinement of Bob, which changes all buffer sizes from 1 to ∞, leading to buffer overflows. Our aim is to overcome this problem by controlling permutations *locally* with the help of the IO-dependency analysis. The key idea is to prohibit the permutation of an output action at k_0 with an input or branching action which prevents (by IO-causality) the accumulation of values in k_0.

Recall Definition 3.4. We define the *minimal resetting paths* to be the paths that satisfy the reset property while none of their suffix does. Then, we define the *dependent nodes* of channel k, noted $\mathtt{dep}(k)$ to be the union of the reset nodes of the minimal resetting paths that end with k. This set of nodes characterises a buffer usage.

First, for a given G, we choose a partition $\{N_0,\ldots,N_n\}$ of the set of nodes of G. This partition should satisfy the two properties: $\forall \mathsf{n} \in N_i, \mathsf{n}' \in N_j, \mathtt{ch}(\mathsf{n}) = \mathtt{ch}(\mathsf{n}') \Rightarrow N_i = N_j$ and $\forall \mathsf{n} \in N_i, \mathtt{dep}(\mathtt{ch}(\mathsf{n})) \subset N_i$. The choice of a partitioning depends in particular on a choice of reset and dependent nodes. Note that the trivial partitioning (with only one partition) is always possible. Since that, for each channel k, all nodes using k are part of the same partition (written $N(k)$), we can annotate all uses of k in G by $N(k)$.

In the example below, the partitioning is made of $N_1 = \{\mathsf{n}_1, \mathsf{n}_2\}$ and $N_2 = \{\mathsf{n}_3, \mathsf{n}_4\}$: we give the annotated session graph (with the IO-dependencies highlighted) on the left and the projected types (where the annotations are kept) on the right.



n1 : Alice → Bob : $s_1^{N_1}$
IO
IO
n2 : Bob → Alice : $s_2^{N_1}$
IO
n4 : Bob → Alice : $s_4^{N_2}$
IO
n3 : Alice → Bob : $s_3^{N_2}$

$$T_{\text{Alice}} = \mu\mathbf{x}.s_1^{N_1}!;s_2^{N_1}?;s_3^{N_2}!;s_4^{N_2}?;\mathbf{x}$$

$$T_{\text{Bob}} = \mu\mathbf{x}.s_1^{N_1}?;s_2^{N_1}!;s_3^{N_2}?;s_4^{N_2}!;\mathbf{x}$$

$$T^{opt}_{\text{Alice}} = \mu\mathbf{x}.s_1^{N_1}!;s_3^{N_2}!;s_2^{N_1}?;s_4^{N_2}?;\mathbf{x}$$

Next, we apply the size-preserving asynchronous communication subtyping, following the annotations on the projected types. The relation $T \ll T'$ means T is more asynchronous than (or more optimised than) T'. The main rule is:

$$\text{(OI)} \qquad k^N!\langle U\rangle;k_0^{N_0}?\langle U'\rangle;T \ \ll\ k_0^{N_0}?\langle U'\rangle;k^N!\langle U\rangle;T \qquad (N \cap N_0 = \emptyset)$$

where the two prefixes are permutable if the IO-chains of the two prefixes are disjoint. We can *always* permute two inputs and two outputs with distinct channels since they do not contribute to the input and output alternations that constitute the IO-chains. The branching/selection rules are similarly defined, and others are context rules. Then we define a coinductive subtyping relation $T_1 \leqslant_c T_2$ as a form of type simulation, following [15,14]. The important fact is that $\leqslant_c$ does not alter buffer sizes: suppose $[\![G]\!] = \{T@\mathtt{p}\}_{\mathtt{p}}$ with $T@\mathtt{p} = (G \upharpoonright \mathtt{p})@\mathtt{p}$ and $\mathtt{p} \in G$. Assume $T@\mathtt{p} \leqslant_c T'@\mathtt{p}$ with $[\![G']\!] = \{T'@\mathtt{p}\}_{\mathtt{p}}$ Then $\mathscr{B}_k\langle G\rangle = \mathscr{B}_k\langle G'\rangle$. Since there is no change in the buffer bounds, Type and Buffer Safety are just proved from Theorem 3.8 and Corollary 3.9.

In the example above, in Alice's type, we can permute $s_2^{N_1}$? and $s_3^{N_2}$! ($T_{\text{Alice}}^{opt} \leqslant_c T_{\text{Alice}}$) since $N_1 \cap N_2 = \emptyset$, keeping the size of each buffer one. Hence process typable by T_{Alice}^{opt} can safely send message at s_3 before input at s_2. In Alice-Bob$_5$-Carol from § 1, the original global type G annotated by IO-chains has only one partition $N = \{\mathsf{n}_1, \mathsf{n}_2, \mathsf{n}_3\}$:

$$\mu\mathbf{x}.\ \text{Alice} \to \text{Bob}\colon s_1^N\,\langle \mathsf{nat}\rangle; \text{Bob} \to \text{Carol}\colon s_2^N\,\langle \mathsf{string}\rangle; \text{Carol} \to \text{Alice}\colon s_3^N\,\langle \mathsf{real}\rangle; \mathbf{x}$$

Bob's local type is $\mu\mathbf{x}.s_1^N?\langle\mathsf{nat}\rangle; s_2^N!\langle\mathsf{string}\rangle; \mathbf{x}$, which prevents any optimisation $\ll$ by (OI). Hence, Bob$_5$ is not typable. Some typable examples are given in the next section.

5.1 Application: Multi-buffering Algorithm

The double buffering algorithm [6] is widely used in high-performance and multicore computing. We generalise this algorithm to *multi-buffering* [16], and solve an open issue in our previous work [15, § 5]. The aim is to transport a large amount of data as a series of units (say each unit is 16kB) from a source (Source) to a transformer (called Kernel). Each unit gets processed at Kernel and delivered to a sink (Sink). Kernel uses n 16kB buffers, named B_i, to maximise the message transfer asynchrony. Processes which represent Source, Sink, Kernel and Optimised Kernel are given below using parameterised processes [22] (i.e. where $\mathtt{foreach}(i<n)\{P[i]\}$ means we iterate $P[i]$ for $0 \le i < n$):

$$\textit{Source:}\ \mu X.\mathtt{foreach}(i<n)\{r_i?();\ s_i!\langle y_i\rangle\};X \qquad \textit{Sink:}\ \mu X.\mathtt{foreach}(i<n)\{t_i!\langle\rangle; u_i?(z_i)\};X$$

$$\textit{Kernel:}\ \mu X.\mathtt{foreach}(i<n)\{r_i!\langle\rangle; s_i?(x_i); t_i?(); u_i!\langle x_i\rangle\};X$$

$$\textit{Optimised Kernel:}\ r_0!\langle\rangle;\ldots;r_{n-1}!\langle\rangle;\mu X.\mathtt{foreach}(i<n)\{s_i?(x_i); t_i?(); u_i!\langle x_i\rangle; r_i!\langle\rangle\};X$$

In the loop, Kernel notifies Source with signals at r_i that it is ready to receive data in each channel s_i of buffer B_i. Source complies, sending one unit via s_i. Then Kernel waits for Sink to inform (via t_i) that Sink is ready to receive data via u_i: upon receiving the signals, Kernel sends the unit of processed data to Sink via u_i. If Kernel sends the n notifications to $r_0,\ldots,r_{n-1}$ *ahead* like Optimised Kernel, Source can start its work for the next unit (sending y_j at s_j) without waiting for other buffers.

The following proposition means that *the n-buffers of size one in Kernel simulate one buffer of size n*, maximising the asynchrony. The proof is done by annotating the global type with partitions $\{r_i, s_i\}$ and $\{u_i, t_i\}$, and checking that the permutation of the projected Kernel type satisfies the (OI) rule.

Proposition 5.1 (n-buffering correctness). *Source-Optimal Kernel-Sink satisfies both progress and communication-safety. Also each buffer at s_i and u_i holds at most one unit.*

If Optimised Kernel is optimised as $r_0!\langle\rangle;\ldots;\mathtt{foreach}(i<n)\{r_i!\langle\rangle; s_i?(x_i); t_i?; u_i!\langle x_i\rangle\}$ (which is not typable in our system), then all buffers are forced to hold 2 *units*. This

unsafe optimisation is typable in [15] but prevented here by Proposition 5.1. In [5], we also deal with a use case of MPSoC buffer allocation from [4], with branching and iterations [22], and verify it by applying all of the previously described methods.

6 Related Work

Checking buffer bounds based on global specifications has been studied through Petri nets and Synchronous data flow. Recent advances [9] in the study of Kahn Process Networks (KPN) have improved Parks's algorithm [18] to ensure safe executions of stream-based applications with bounded buffers, using an appropriate scheduling policy. Their theory is applied to KPN applications on MPSoC [4], demonstrating the effectiveness of non-uniform, fine-grained buffer allocations. By contrast, our approach is type-based and relies on the existence of a global specification that brings additional guarantees (such as deadlock-freedom) and allows global choreography manipulation and refinements. It is moreover directly applicable to programming languages [12,22] by extending existing type syntax and checking.

The idea of using a type-abstraction to investigate channel communications goes back to Nielson & Nielson's work on CML [17]. Gay & Vasconcelos [8] propose a linear type system for binary sessions to enforce buffer bounds computed by a fixed point method. Their work is thus limited to a particular channel allocation (i.e. principal, cf. § 4) and does not extend to multiparty interactions (their method would find that the buffers in Example (a) are infinite). Terauchi & Megacz [21] describe a polynomial method to infer buffer bounds of a concurrent language through program analysis using linear programming techniques, improving on previous work in [13], see [21, § 7]. Our bound computation method differs in that it starts from a direct type-based abstraction of global interaction structures, namely session graphs, not from direct investigation of local types nor processes (normally in distributed systems, a peer does not know other peer's type or implementation [12]). It also leads to the general simplicity of the analysis, and the uniform treatment of subtle issues such as asynchronous optimisations. Thanks to session types, the channel passing problem in [21, § 6] does not arise in our analysis: different (possibly newly generated) sessions and names can be stored in the same buffer, still giving the exact bound of stored channels. None of [8,21] have studied either channel allocation, global refinement or messaging optimisation.

Among process calculi for service-oriented computing (SOC), contracts [3] and the conversation calculus [2] provide static type checking for a series of interactions and ensure progress. We demonstrate the advantage of global types by the simplicity of our analysis and the uniform treatments and articulation of our various algorithms. Our approach is, however, extensible to these calculi because (1) the IO-causality analysis does not rely on the form of session branches so that other form of sums can be analysed by the same technique; and (2) combining with a polynomial inference which builds a graph from a collection of local types $[\![G]\!]$ [15], Subject Reduction Theorem can be proved using our invariance method noting that we use $[\![G]\!]$ for the proofs. An extension to other formalisms for SOC including [3,2] is an interesting future work.

Further topics include the enrichment of global types with more quantitative information (such as distance, probabilities and weights), which would enable finer-grained analyses and optimisations.

References

1. Bettini, L., Coppo, M., D'Antoni, L., Luca, M.D., Dezani-Ciancaglini, M., Yoshida, N.: Global progress in dynamically interleaved multiparty sessions. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 418–433. Springer, Heidelberg (2008)
2. Caires, L., Vieira, H.T.: Conversation types. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 285–300. Springer, Heidelberg (2009)
3. Castagna, G., Padovani, L.: Contracts for mobile processes. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 211–228. Springer, Heidelberg (2009)
4. Cheung, E., Hsieh, H., Balarin, F.: Automatic buffer sizing for rate-constrained KPN applications on multiprocessor system-on-chip. In: HLDVT 2007, pp. 37–44. IEEE, Los Alamitos (2007)
5. Deniélou, P.-M., Yoshida, N.: Buffered communication analysis in distributed multiparty sessions, Full version, Prototype at, http://www.doc.ic.ac.uk/~pmalo/multianalysis
6. Donaldson, A., Kroening, D., Rümmer, P.: Automatic analysis of scratch-pad memory code for heterogeneous multicore processors. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 280–295. Springer, Heidelberg (2010)
7. Fähndrich, M., et al.: Language support for fast and reliable message-based communication in singularity OS. In: EuroSys 2006, ACM SIGOPS, pp. 177–190. ACM Press, New York (2006)
8. Gay, S., Vasconcelos, V.T.: Linear type theory for asynchronous session types. In: JFP (2009)
9. Geilen, M., Basten, T.: Requirements on the Execution of Kahn Process Networks. In: Degano, P. (ed.) ESOP 2003. LNCS, vol. 2618, pp. 319–334. Springer, Heidelberg (2003)
10. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type disciplines for structured communication-based programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998)
11. Honda, K., Yoshida, N., Carbone, M.: Multiparty Asynchronous Session Types. In: POPL, pp. 273–284 (2008)
12. Hu, R., Yoshida, N., Honda, K.: Session-Based Distributed Programming in Java. In: Vitek, J. (ed.) ECOOP 2008. LNCS, vol. 5142, pp. 516–541. Springer, Heidelberg (2008)
13. Kobayashi, N., Nakade, M., Yonezawa, A.: Static analysis of communication for asynchronous concurrent programming languages. In: Mycroft, A. (ed.) SAS 1995. LNCS, vol. 983, pp. 225–242. Springer, Heidelberg (1995)
14. Mostrous, D., Yoshida, N.: Session-Based Communication Optimisation for Higher-Order Mobile Processes. In: Curien, P.-L. (ed.) TLCA 2009. LNCS, vol. 5608, pp. 203–218. Springer, Heidelberg (2009)
15. Mostrous, D., Yoshida, N., Honda, K.: Global principal typing in partially commutative asynchronous sessions. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 316–332. Springer, Heidelberg (2009)
16. Multi-buffering, http://en.wikipedia.org/wiki/Multiple_buffering
17. Nielson, H., Nielson, F.: Higher-order concurrent programs with finite communication topology. In: POPL, pp. 84–97 (1994)
18. Parks, T.: Bounded Scheduling of Process Networks. PhD thesis, California Barkeley (1995)
19. Rota, G.-C.: The number of partitions of a set. Amer. Math. Monthly 71, 498–504 (1964)
20. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: Halatsis, C., Philokyprou, G., Maritsas, D., Theodoridis, S. (eds.) PARLE 1994. LNCS, vol. 817, pp. 398–413. Springer, Heidelberg (1994)
21. Terauchi, T., Megacz, A.: Inferring channel buffer bounds via linear programming. In: Drossopoulou, S. (ed.) ESOP 2008. LNCS, vol. 4960, pp. 284–298. Springer, Heidelberg (2008)
22. Yoshida, N., Deniélou, P.-M., Bejleri, A., Hu, R.: Parameterised multiparty session types. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 128–145. Springer, Heidelberg (2010)

Efficient Bisimilarities from Second-Order Reaction Semantics for π-Calculus*

Pietro Di Gianantonio[1], Svetlana Jakšić[2], and Marina Lenisa[1]

[1] Dipartimento di Matematica e Informatica, Università di Udine, Italy
{digianantonio,lenisa}@dimi.uniud.it
[2] Faculty of Engineering, University of Novi Sad, Serbia
sjaksic@uns.ac.rs

Abstract. We investigate Leifer and Milner *RPO approach* for deriving efficient (finitely branching) LTS's and bisimilarities for π-calculus. To this aim, we work in a category of *second-order term contexts* and we apply a general *pruning technique*, which allows to simplify the set of transitions in the LTS obtained from the original RPO approach. The resulting LTS and bisimilarity provide an alternative presentation of *symbolic LTS* and Sangiorgi's *open bisimilarity.*

Introduction

Recently, much attention has been devoted to deriving *labelled transition systems* and *bisimilarity congruences* from *reactive systems*, in the context of process languages and graph rewriting. Through the notion of *contextual equivalence*, reactive systems naturally induce behavioural equivalences which are *congruences* w.r.t. contexts, while LTS's naturally induce *bisimilarity equivalences* with coinductive characterizations. However, such equivalences are not congruences in general, and it can be a difficult task to derive LTS's inducing bisimilarities which are congruences.

Leifer and Milner [1] presented a general categorical method, based on the notion of Relative Pushout (RPO), for deriving a transition system from a reactive system, in such a way that the induced *bisimilarity* is a *congruence*. The labels in Leifer-Milner's transition system are those *contexts* which are *minimal* for a given reaction to fire. In the literature, some case studies have been carried out in the setting of process calculi, for testing the expressivity of Leifer-Milner's approach [2,3,4,5,6,7,8]. Moreover, to deal with structural rules, an elaboration of the RPO theory in the *G-category setting* (GRPO) has been introduced by Sassone and Sobocinski in [2].

In general, in applying the RPO construction one needs to deal with the following problems:
– To encode all the characteristics of the language, mainly: structural rules, name abstraction, name hiding.

* Work partially supported by PRIN Project SISTER 20088HXMYN and FIRB Project RBIN04M8S8, both funded by MIUR.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 358–372, 2010.

– To obtain a label transition system which is usable, where proofs of bisimilarities require to consider only a *finite* set of transitions at each step. Almost always, the RPO approach generates LTS's that are quite large and often redundant, in the sense that most of the transitions can be eliminated from the LTS without affecting the induced bisimilarity.
– When the RPO construction is performed, by embedding the category of terms in a larger category, the resulting LTS can contain states that do not correspond to any term of the language, and whose intuitive meaning is difficult to grasp.

In order to solve the above problems, the RPO construction needs to be tuned-up, that is we have to find a convenient category in which to perform the construction, and general methods for pruning the LTS.

In a previous work [7], we solve the above problems for the prototypical example of CCS. In [7], we use a *category of term contexts*, *i.e.* a Lawvere category. We encode names, and name binding using *de Bruijn indexes*; this allows a relatively simple and formally correct treatment of names, which, when represented natively, can be quite subtle to treat. Moreover, in [7] we introduce a general technique, which allows to *prune* an LTS obtained from a RPO-construction, without modifying the induced bisimilarity. This is achieved by eliminating *definable* sets of transitions, *i.e* transitions whose effect can be obtained by other transitions. In [7], by using the above ideas in applying the (G)RPO construction to CCS, we obtain the *standard LTS* from the *standard reaction semantics*. This is an indication that the (G)RPO technique in combination with our general pruning technique can generate useful LTS's.

In the present work, we treat in detail the π-calculus. The techniques developed for CCS turn out to be useful also for the π-calculus, but for the latter, in order to get an efficient LTS, a further ingredient is necessary, *i.e. second-order contexts*. Categories *of second-order term contexts* have been introduced in [9] as generalizations of the Lawvere category of terms, where *parametric rules* can be readily represented. Intuitively, if we apply Leifer-Milner technique to π-calculus by working in a standard Lawvere category of term contexts, in the resulting LTS, for any process P exposing an output prefix, we need to consider transitions $P \xrightarrow{[\,]|a(x).Q} P'$, for all Q. All these label contexts are "minimal" for the reduction to fire; we cannot avoid Q, since, in the resulting process P', a substitution is applied to Q. This makes the LTS inefficient. To overcome this problem, we use second-order contexts. In this way, all the above transitions can be parametrically captured by a single transition $P \xrightarrow{[\,]|a(x).X} P''$, where X is a variable representing a generic term, which will be possibly instantiated in the future.

The final result of our construction produces a bisimilarity which is a mild variation of Sangiorgi's *open bisimilarity*. In order to get the final efficient characterization of our bisimilarity, we need a further ad-hoc pruning. However, even if the GRPO construction does not directly give the final result, once applied, it produces an LTS which is a superset of the final usable one. Identifying redundant transitions is then not so difficult; the only difficult part is to prove that these are redundant.

Interestingly enough, our analysis provides new insights on the theory of π-calculus, namely we obtain an alternative presentation of symbolic LTS and open bisimilarity, where *distinctions* do not appear.

Remarkably, the Leifer-Milner technique has lead us to a bisimilarity congruence substantially in a direct way, just using general tools, without the need of new concepts. Whereas, in the standard treatment, in moving from CCS to π-calculus, various new notions are required, such as bound output transitions, distinctions, etc. In conclusion, the results for CCS of [7] and the above results for π-calculus are rather satisfactory, and they are an indication that the general techniques used in this paper could also give new insights on more recent calculi, whose theory is still evolving.

Related Work. The RPO construction has been applied to π-calculus in [3,10]. In [3], History Dependent Automata are used to present a reactive system for the fragment of π-calculus without the ν-operator. The reactive system is obtained by starting from an LTS and then incorporating the labels in the initial state of the transition. The reactive system considered in [10] is based on the theory of bigraphs and models the asynchronous π-calculus.

The present work is also related to [11]. Both works use categories that are suitable generalizations of the Lawvere category of contexts. However, in our work we strictly apply the RPO construction to derive an LTS for the π-calculus, while [11] uses the RPO construction as a sort of inspiration for defining directly an LTS for the π-calculus. The two works use a quite different notion of generalized context, and thus also the obtained LTS's are quite different.

Summary. In Section 1, a presentation of π-calculus syntax with de Bruijn indexes and parametric reaction semantics is given. In Section 2, the GRPO technique is applied to π-calculus, and efficient characterizations of the GIPO bisimilarity are investigated. In Section 3, GIPO bisimilarity is compared with open bisimilarity. Final remarks appear in Section 4. In the extended version of the present paper [12], the theory of RPO's in the G-category setting and the general pruning technique of [7] are recalled, and some proofs are presented.

1 Second-Order π-Calculus Processes

In this section, we present a version of π-calculus with *de Bruijn indexes* together with reaction semantics. Such presentation allows us to deal smoothly with binding operators, and it is needed for extending to contexts the structural congruence on processes. In our presentation, π-calculus *names* $a_0, a_1, \dots$ are replaced by de Bruijn indexes $r_0, r_1, \dots$, which are *name references.*

Intuitively, a name reference can be viewed as a link (or a pointer). So a bound name is replaced by a link to the corresponding binding operator, while a free name is replaced by a link to its occurrence in a list of names. Concretely, links are represented by natural numbers, and:

- binding operators ν and input prefix do not contain any name;
- the index r_i refers to the free name a_j if $j = i - n \geq 0$ and r_i appears under the scope of n binding operators;

- otherwise, if $i < n$, then r_i is bound by the $i+1$-th binding operator on its left. *E.g.* in $\nu r_1().\overline{r_2}r_0.0$, r_0 is bound by the input prefix $r_1()$, while r_1 and r_2 both refer to the free name a_0. In standard syntax, the above process will be written as $(\nu a)a_0(a').\overline{a_0}a'.0$.

Definition 1 (π-calculus Processes). *Let $r_0, r_1, \ldots \in \mathcal{R}$ be* name references*; we will use r, s as metavariables for name references. We define*

$$\begin{array}{rl} (\mathcal{A}ct \ni)\ \alpha ::= \tau \ \mid\ r() \ \mid\ \overline{r}s & \text{actions} \\ (\mathcal{G} \ni)\ M ::= 0 \ \mid\ M_1 + M_2 \mid \alpha.P \ \mid\ Y & \text{guarded processes} \\ (\mathcal{P} \ni)\ P ::= M \mid X \mid \nu P \mid P_1|P_2 \mid rec\ X.P \mid \sigma P & \text{processes} \end{array}$$

where
- $X, X_0, X_1, \ldots \in \mathcal{X}$ are process variables, and $Y, Y_0, Y_1, \ldots \in \mathcal{Y}$ are guarded process variables; we will use Z to range over $\mathcal{X} \cup \mathcal{Y}$;
- the process variable X appears guarded in rec $X.P$;
- σ is a name substitution obtained as a finite composition of the transformations $\{\delta_i\}_{i\geq 0} \cup \{s_i\}_{i\geq 0} \cup \{t_{ij}\}_{i,j\geq 0}$, where δ_i, s_i represent the i-th shifting and the i-th swapping, respectively, and $t_{i,j}$ are the singleton substitutions, defined by:

$$\delta_i(r_j) = \begin{cases} r_{j+1} & \text{if } j \geq i \\ r_j & \text{if } j < i \end{cases} \qquad s_i(r_j) = \begin{cases} r_j & \text{if } j \neq i, i+1 \\ r_{i+1} & \text{if } j = i \\ r_i & \text{if } j = i+1 \end{cases}$$

$$t_{i,j}(r_k) = \begin{cases} r_k & \text{if } k \neq i \\ r_j & \text{if } k = i \end{cases}$$

A closed process *is a process in which each occurrence of a variable is in the scope of a rec operator.*

In the following definition, we introduce the notion of *second-order context*, consisting of a *variable substitution θ* and a *first-order context*:

Definition 2 (Second-order Contexts). *We define the* second-order 1-hole process contexts (contexts) *by:*

$$\mathbb{C} ::= \ [\ ]_\theta \mid \nu\mathbb{C} \mid P + \mathbb{C} \mid \mathbb{C} + P \mid P|\mathbb{C} \mid \mathbb{C}|P \mid rec\ X.\mathbb{C} \mid \sigma\mathbb{C}$$

where $\theta = \theta^{\mathcal{X}} + \theta^{\mathcal{Y}} : \mathcal{X} + \mathcal{Y} \to \mathcal{P} + \mathcal{G}$ is a substitution of processes for process variables, mapping (guarded) process variables into (guarded) processes.

Notation. *We will often denote substitutions by the list of variables which are actually substituted,* i.e. *as $\{P_1/X_1, \ldots, P_m/X_m, M_1/Y_1, \ldots, M_n/Y_n\}$, omitting the variables which are left unchanged. Moreover, for denoting second-order contexts, we will also use the notation $C[\ ]_\theta$, when we need to make explicit the variable substitution θ.*

Notice that in the above definition of contexts we do not distinguish between guarded and general contexts, thus also “ill-formed” contexts, such as $([\ ]_\theta|P)+P'$ are included at this stage. In Section 2, where we will apply the GIPO technique, we will give a precise definition of guarded and general contexts.

In what follows, we will refer to π-calculus processes with de Bruijn indexes and second-order contexts as *terms*, denoted by T. Intuitively, when a second-order context $C[\ ]_\theta$ is applied to a term T, the variable substitution θ is applied to T and the resulting term is put in the hole. In order to formalize this notion of context application, we first need to introduce the notion of applying a substitution to a term:

Definition 3 (Context Application)
(i) Let T be a term, and let θ be a variable substitution. We define the extension $\widehat{\theta}$ to terms, by induction on T as:

$$\widehat{\theta}(Z) = \theta(Z) \qquad \widehat{\theta}([\ ]_{\theta'}) = [\ ]_{\widehat{\theta}\circ\theta'}$$
$$\widehat{\theta}(T_1 + T_2) = \widehat{\theta}(T_1) + \widehat{\theta}(T_2) \qquad \widehat{\theta}(T_1 \mid T_2) = \widehat{\theta}(T_1) \mid \widehat{\theta}(T_2)$$
$$\widehat{\theta}(\sigma T) = \sigma\widehat{\theta}(T) \qquad \widehat{\theta}(\nu T) = \nu(\widehat{\theta}(T))$$
$$\widehat{\theta}(rec\ X.T) = rec\ X.\widehat{\theta}'(T)\ , \qquad where\ \theta'(Z) = \begin{cases} \theta(Z) & if\ Z \neq X \\ X & if\ Z = X \end{cases}$$

In what follows, by abuse of notation, we will often denote $\widehat{\theta}(T)$ simply by $\theta(T)$.
(ii) Let $\mathbb{C}$ be a context and let T be a term, the application of $\mathbb{C}$ to T, denoted by $\mathbb{C} \cdot T$, is defined by induction on $\mathbb{C}$ by:

$$[\ ]_\theta \cdot T = \widehat{\theta}(T) \qquad \nu\mathbb{C} \cdot T = \nu(\mathbb{C} \cdot T)$$
$$(P + \mathbb{C}) \cdot T = P + (\mathbb{C} \cdot T) \qquad (\mathbb{C} + P) \cdot T = (\mathbb{C} \cdot T) + P$$
$$(P \mid \mathbb{C}) \cdot T = P \mid (\mathbb{C} \cdot T) \qquad (\mathbb{C} \mid P) \cdot T = (\mathbb{C} \cdot T) \mid P$$
$$(rec\ X.\mathbb{C} \cdot T) = rec\ X.(\mathbb{C} \cdot T) \qquad (\sigma\mathbb{C}) \cdot T = \sigma(\mathbb{C} \cdot T)$$

In order to apply the GRPO technique to π-calculus, it is convenient to extend the structural congruence, which is usually defined only on processes, to all contexts. Here is where the syntax presentation à la de Bruijn plays an important rôle. Namely the π-calculus rule

$$(\nu aP) \mid Q \equiv \nu a(P \mid Q)\ ,\ \text{if } a \text{ not free in } Q$$

is problematic to extend to contexts with the usual syntax, since, if Q is a context, we have to avoid captures by the ν-operator of the free variables of the processes that will appear in the holes of Q. Using de Bruijn indexes (and index transformations), the above rule can be naturally extended to contexts as:

$$(\nu P) \mid \mathbb{C} \equiv \nu(P \mid \delta_0\mathbb{C})$$

where the shifting operator δ_0 avoids the capture of free name references. In the standard syntax there is no way of defining a general name substitution playing the role of δ_0.

The complete definition of the structural congruence is as follows:

Definition 4 (Structural Congruence). *Let T be a term.* Structural congruence *is the equivalence relation $\equiv$, closed under process constructors, inductively generated by the usual axioms on $|$, $+$, and by:*

$$\begin{array}{ll}
(\mathbf{nu}) & \nu 0 \equiv 0 \quad T|(\nu T') \equiv \nu((\delta_0 T)|T') \quad \nu\nu T \equiv \nu\nu \mathbf{s}_0 T \\
 & \tau.\nu P \equiv \nu\tau.P \quad \overline{r}s.\nu P \equiv \nu\delta_0(\overline{r}s).P \quad r().\nu P \equiv \nu\delta_0(r()).s_0 P
\end{array}$$

$$\begin{array}{lll}
(\mathbf{sigma}) & \sigma 0 \equiv 0 & \sigma(\overline{r}s.T) \equiv \overline{\sigma(r)}\sigma(s).\sigma(T) \\
 & \sigma(\tau.T) \equiv \tau.\sigma(T) & \sigma(r().T) \equiv \sigma(r)().\sigma_{+1}T \\
 & \sigma(T|T') \equiv \sigma(T)|\sigma(T') & \sigma(rec\ X.T) \equiv rec\ X.(\sigma T) \\
 & \sigma(T+T') \equiv \sigma(T)+\sigma(T') & \sigma(\nu T) \equiv \nu(\sigma_{+1}T) \\
 & \sigma_1 \ldots \sigma_m T \equiv \sigma'_1 \ldots \sigma'_n T\ , & \mathit{if}\ \ \sigma_1 \circ \ldots \circ \sigma_m = \sigma'_1 \circ \ldots \circ \sigma'_n
\end{array}$$

$$(\mathbf{subs}) \quad [\ ]_\theta \equiv [\ ]_{\theta_1}\ \mathit{if}\ \forall X\ \ \theta(X) \equiv \theta_1(X) \qquad (\mathbf{rec}) \quad rec\ X.P \equiv P[rec\ X.P/X]$$

$$\mathit{where}\ \sigma_{+1}(r_i) = \begin{cases} r_0 & \mathit{if}\ i = 0 \\ (\sigma(r_{i-1}))_{+1} & \mathit{otherwise} \end{cases} \qquad \sigma(\alpha) = \begin{cases} \overline{\sigma}(r) & \mathit{if}\ \alpha = \overline{r} \\ \sigma(r) & \mathit{if}\ \alpha = r \\ \tau & \mathit{if}\ \alpha = \tau \end{cases}$$

The last three (**nu**)-rules are not standard in π-calculus presentations, since they are not strictly necessary for proving the basic syntactic properties of the calculus. However, they are safe because they allow to move, inside/outside the ν operator, prefixes which are not captured by ν, see *e.g.* [13]. The assumption of such extra rules is not essential in our presentation, however it simplifies the GIPO construction. As far as the (**sigma**)-rule, notice that there is an effective procedure to determine whether $\sigma_1 \circ \ldots \circ \sigma_m = \sigma'_1 \circ \ldots \circ \sigma'_n$. Namely, the two compositions are equal if and only if they contain the same number of transformations in the forms δ_i and their behaviour coincides on an initial segment of indexes (whose length can be calculated from the δ_i's and the s_i's involved). Finally, the unfolding rule (**rec**) is given only for processes P. It cannot be directly extended to contexts, since their unfolding can produce multiple-hole contexts. However, the above (**rec**)-rule is sufficient for our purposes, since we will only need it in reducing processes.

As in the standard presentation, one can easily show that each π-calculus process P is structurally congruent to a process in *normal form*, *i.e.* a process of the shape $\nu^k(\Sigma_{j=1}^{m_1} S_{1,j} \mid \ldots \mid \Sigma_{j=1}^{m_n} S_{n,j})$, where all unguarded restrictions are at the top level, and name substitutions do not appear at the top level. We use S to range over processes of the shape $\alpha.P$ or σY. If $m_i = 1$ for some $i \in \{1, \ldots n\}$ then S can also be of the form σX.

Definition 5 (Reaction Semantics). *The* reaction relation $\rightarrow$ *is the least relation closed under the following* reaction rules *and* reaction contexts*:*

Reaction rules.

$$(r().X_1 + Y_1) \mid (\overline{r}r_j.X_2 + Y_2) \rightarrow (\nu(t_{0,j+1}X_1)) \mid X_2$$

$$\tau.X + Y \rightarrow X$$

Reaction contexts. $\mathbb{D} ::= \ [\ ]_\theta \mid \nu\mathbb{D} \mid P|\mathbb{D} \mid \mathbb{D}|P \mid \sigma\mathbb{D}$

where σ is a permutation of name references (a one to one reference substitution).

Notice that the permutation σ in the definition of reaction contexts is not strictly necessary for defining the reaction semantics. It could be omitted, without changing the reaction semantics, since, using the congruence rules, name substitutions distribute over the actions. However, in view of the GIPO construction of Section 2 it is useful to include it.

A mapping $\mathcal{T}$ from standard π-calculus syntax into our de Bruijn presentation can be defined by structural induction, using an extra set of names with negative indexes $(a_{-1}, a_{-2}, \ldots)$. The most meaningful cases are: $\mathcal{T}(P) = \mathcal{T}_0(P)$, $\mathcal{T}_n(a_i(a_j).P) = r_{i+n}().\mathcal{T}_{n+1}(P_{\{a_{-n-1}/a_j\}})$, $\mathcal{T}_n(\overline{a_i}a_j.P) = \overline{r_{i+n}}r_{j+n}.\mathcal{T}_n(P)$.

For any pair of π-calculus processes P, Q on the standard syntax, it turns out that $P \to Q$ in the ordinary reaction system iff $\mathcal{T}(P) \to \mathcal{T}(Q)$ in our reaction system. We omit the details.

2 Applying the GIPO Technique to Second-Order π-Calculus

For lack of space, we do not present in detail the (G)RPO construction, we refer to [1] for a general introduction to the RPO technique, to [2] for the presentation of the GRPO technique and to [7] or to [12], for a compact presentation of all the theory on which the results presented here are based.

However, in order to grasp most of the material presented in the paper, the following informal and intuitive explanations of the GRPO construction may suffice. The main idea in the RPO construction is to define an LTS, starting from a reaction system. The states of the derived LTS are terms, while the labels are the *minimal contexts* necessary for a given reaction to fire. In more detail, the LTS contains the transition $t \xrightarrow{\mathbb{C}}_I v$, if the reaction system contains the reaction $\mathbb{C} \circ t \to v$, and for no subcontext $\mathbb{C}'$ of $\mathbb{C}$ and no subterm v' of v, there is a reaction $\mathbb{C}' \circ t \to v'$. This idea is formalized using a category where arrows represent terms or contexts. The notion of minimal context is defined in terms of a (relative) pushout construction. The main theoretical result is that the LTS, obtained by the RPO construction, induces a *bisimilarity* that is a *congruence*. The GRPO technique is a further elaboration of the RPO technique necessary to deal with the structural rules of the syntax; here the main idea is to perform the RPO construction in a *2-category*. A 2-category is a category having an extra notion of morphism between arrows. When such morphisms are isomorphisms, as in the GRPO construction, the 2-category is called *G-category*. In our setting, morphisms between two arrows represent a structural congruence between two terms (the two arrows), together with an induced mapping between occurrences of name references in the two terms. G-categories always allow to distinguish between two different name references denoting the same name, also when structural rules are used. In some cases, the RPO construction in the standard categories having as arrows equivalence classes of terms fails to produce the correct transitions, an example being $r_0().0 \mid \overline{r_0}r_1.0$, see [2] for more details.

We define here the G-category formed by the *finite* (*i.e.* without the *rec* operator) second-order π-calculus terms equipped with structural congruence. We restrict the G-category to contain only finite processes, because we need the 2-cell morphisms to be isomorphisms. When π-calculus processes contain the *rec* operator, two congruent processes can contain different numbers of actions, so, in general, there does not exist a one-to-one map between occurrences of name references. It is possible to recover an LTS for the whole set of π-processes by extending the rules obtained for the finite calculus, namely allowing parametric rules to be applied also to terms containing the *rec* operator (and by considering the unfolding rule for *rec*). Quite general arguments, based on the notion of finite approximation, show that, in the extended LTS, the bisimilarity is still a congruence. Moreover, once restricted to finite processes, in the definition of π-calculus term category, it is sufficient to consider *linear* terms, that is terms where each variable appears at most once. This restriction is justified by the fact that, in the GIPO transition system, closed terms generate only linear open terms; moreover, it simplifies the GIPO construction below.

Since the π-calculus grammar needs to distinguish between guarded and generic terms, the category needs to contain two corresponding distinct objects. Formally:

Definition 6 (Category of Second-order π-calculus Terms). *Let $\mathcal{C}_\pi$ be the category defined by:*
- Objects are ϵ, $\mathcal{G}$, $\mathcal{P}$.
- Arrows from ϵ to $\mathcal{G}$ ($\mathcal{P}$) are linear (un)guarded processes, i.e. *processes where each variable appears at most once. Arrows $\mathcal{A} \to \mathcal{B}$ are the contexts $\mathbb{C}^{\mathcal{B}}_{\mathcal{A}}$ generated by the grammar:*

$$\begin{aligned}
&\mathbb{C}^{\mathcal{G}}_{\mathcal{G}} ::= [\,]_\theta \mid \alpha.\mathbb{C}^{\mathcal{P}}_{\mathcal{G}} \mid \ \mathbb{C}^{\mathcal{G}}_{\mathcal{G}} + M \mid M + \mathbb{C}^{\mathcal{G}}_{\mathcal{G}}\\
&\mathbb{C}^{\mathcal{G}}_{\mathcal{P}} ::= \alpha.\mathbb{C}^{\mathcal{P}}_{\mathcal{P}} \mid \mathbb{C}^{\mathcal{G}}_{\mathcal{P}} + M \mid M + \mathbb{C}^{\mathcal{G}}_{\mathcal{P}}\\
&\mathbb{C}^{\mathcal{P}}_{\mathcal{G}} ::= \mathbb{C}^{\mathcal{G}}_{\mathcal{G}} \mid \nu\mathbb{C}^{\mathcal{P}}_{\mathcal{G}} \mid \mathbb{C}^{\mathcal{P}}_{\mathcal{G}}|P \mid P|\mathbb{C}^{\mathcal{P}}_{\mathcal{G}} \mid \sigma\mathbb{C}^{\mathcal{P}}_{\mathcal{G}}\\
&\mathbb{C}^{\mathcal{P}}_{\mathcal{P}} ::= [\,]_\theta \mid \mathbb{C}^{\mathcal{G}}_{\mathcal{P}} \mid \nu\mathbb{C}^{\mathcal{P}}_{\mathcal{P}} \mid \mathbb{C}^{\mathcal{P}}_{\mathcal{P}}|P \mid P|\mathbb{C}^{\mathcal{P}}_{\mathcal{P}} \mid \sigma\mathbb{C}^{\mathcal{P}}_{\mathcal{P}}
\end{aligned}$$

where any context $\mathbb{C}^{\mathcal{B}}_{\mathcal{A}} = C[\,]_\theta$ is linear, i.e. *any variable appears at most once in $C[\,]$ and in the codomain of θ.*
The identity arrow on $\mathcal{G}$ and $\mathcal{P}$ is $[\,]_{id}$. The only arrow with codomain ϵ is the identity. The composition between morphisms $T : \mathcal{A} \to \mathcal{A}'$, $T' : \mathcal{A}' \to \mathcal{A}''$ is the context application $T' \cdot T$.

In what follows, when not necessary, we will omit tags from contexts. One can easily prove that the above definition is well-posed. In particular, associativity of composition follows from associativity of composition of variable substitutions.

By induction on a proof of structural congruence, it is possible to show that two structurally congruent finite terms have the same number of occurrences for each action, and each proof of congruence induces a one to one map between instances of name references in an obvious way. Thus we can define:

Definition 7 (2-cell isomorphisms). *2-cell isomorphisms between T and T' in $\mathcal{C}_\pi$ are the one-to-one maps between occurrences of name references in T and T', induced by the proof of structural congruence.*

The above maps induce a structure of G-category on $\mathcal{C}_\pi$. Horizontal composition corresponds to the union of the one-to-one maps , while vertical composition amounts to standard function composition. One can easily check that horizontal and vertical compositions are well-behaved, in particular the "middle-four interchange law" holds. Thus we have:

Proposition 1. *The structural congruence on terms induces a structure of G-category on $\mathcal{C}_\pi$.*

Now we can define the G-reaction system of finite (second order) π-calculus processes:

Definition 8 (G-reaction system). *The G-reaction system $\mathbf{C}_\pi$ consists of*
- the G-category of π-calculus terms $\mathcal{C}_\pi$;
- the distinguished object ϵ;
- the subset of linear reaction contexts of Definition 5;
- the reaction rules of Definition 5.

One can easily check that the set of reaction contexts as defined above are composition-reflecting and closed under 2-cells. In particular, in proving that contexts are composition-reflecting, it turns out to be essential to have included also reaction contexts of the shape $\sigma\mathbb{D}$, for σ a permutation.

Proposition 2. *The G-reaction system $\mathbf{C}_\pi$ has redex GRPOs.*

Table 1 summarizes the GIPO contexts (i.e. the labels in the derived LTS) for every possible term (up-to structural congruence). For simplicity, we denote a term equivalence class simply by a special representative. For each process P, on the basis of its form (specified in the first column of the table), the corresponding GIPO contexts are listed, i.e. the "minimal" contexts which make possible a certain reaction. Redex squares can be classified according to the following "parameters":

- type of the reaction rule (τ-reaction or communication);
- how elements of the redex are obtained: (1) already present in P, (2) by instantiating variables in P, (3) appearing in the context;
- in case of variable instantiation by an output action, the name sent can be either private or public. A detailed description of the GIPO contexts of Table 1 and a proof of the above proposition appear in [12].

The GIPO LTS described in Table 1 is quite redundant. Namely, there are many GIPO contexts which are intuitively redundant; *e.g.* all contexts in rows 3 and 13, which are "not engaged". Moreover, in various other cases the effect of some GIPO contexts can be simulated by a restricted set of simpler contexts. Many redundant contexts can be eliminated by applying the general pruning technique presented in [7]. The result is the LTS of *reduced GIPO contexts*, R, formed by the contexts marked by $*$ in the column R of Table 1, in which the name substitution β is restricted to be the identity. Namely, the GIPO LTS of Table 1 is *definable* from the set R of reduced GIPO contexts. A proof of this can be found in [12]. As a consequence, our general pruning technique ensures that the bisimilarity $\sim_R$ induced by the LTS defined in column R coincides with the original GIPO bisimilarity $\sim_G$, and hence it is a congruence.

A further simplified LTS can be obtained by an ad-hoc analysis. We define an LTS, F, composed by the GIPO contexts marked by $\star$ in Table 1. The proof that the bisimilarity induced by the LTS F coincides with the GIPO bisimilarity is based on the technique of the "bisimulation up-to", and it appears in [12].

Proposition 3. *The bisimilarity $\sim_F$ induced by the LTS F coincides with the original GIPO bisimilarity $\sim_G$, and hence it is a congruence.*

Apparently, the LTS F obtained is still infinitely branching. This is due to the fact that we consider transitions where the context contains an output action $\overline{r}s.X$, and s can be any reference. But, when comparing two processes P, Q in the bisimilarity relation, it is sufficient to consider s to be a reference to a name in P or Q, or a reference to just a new name not appearing in P or Q. In this way, we get a finitely branching LTS.

Now, if our aim is to define a bisimilarity relation on π-calculus processes which do not contain process variables, then it is possible to consider a much simpler LTS, namely the LTS of Table 2. This LTS is intended for processes in the form $\nu^k(P' \mid \sigma X)$, with P' a closed process. The above set of processes is closed by all transitions, but 5, which is then meant to be applied just once. Intuitively, X plays the rôle of the environment, that can send to or receive names from P'; the name substitution σ records the names received from P'.

Here we present a detailed description of transitions in Table 2. Row 1 corresponds to a τ-reaction. Row 2 corresponds to the case where the process P exposes two complementary actions. In this rule ι is the identity, if the channel references r and r' in the complementary actions already matches, or a singleton substitution fusing the two channel references, otherwise. Here we use a function $[\![\ ,\ ]\!]$ to express the fact that the two *occurrences* of name references in the complementary actions refer to the same name. This function, given a process and an *occurrence* of a name reference r_i in it, provides the "absolute" index of the name referred by the the occurrence r_i, if r_i is free in P, that is $[\![P, r_i]\!] = j$ means that r_i refers to the free name a_j; otherwise, if r_i is bound, $[\![P, r_i]\!]$ provides the negative index corresponding to the nesting level of the occurrence r_i inside the binding operators (ν or input prefix) in P (we omit the formal definition). Rows 3 and 4 take into account the case where the process P exposes either an input or an output action and the GIPO context provides the complementary action for the communication. In these rules the variable substitution δ sends all variables into variables with even index (see the note at the bottom of Table 1), and it is used to preserve linearity in the term $\mathbb{C} \cdot P$. Namely, δ ensures that the variables with odd indexes will not appear in the process, and hence they can be used in the context. In row 5 the GIPO context instantiates the variable X by the whole communication redex.

In order to compare two closed processes P, Q, we proceed by comparing the processes $P|X$ and $Q|X$, using the LTS of Table 2. Namely, if $\sim_C$ denotes the induced bisimilarity, we have:

Proposition 4. *For any pair of closed processes P, Q, we have that $P \sim_F Q$ iff $P \mid X \sim_C Q \mid X$.*

Table 1. π-calculus GIPO contexts

	Process $P \equiv \nu^k(\Sigma_{j=1}^{m_1} S_{1,j} \mid \ldots \mid \Sigma_{j=1}^{m_n} S_{n,j})$	**GIPO context** $\mathbb{C}$	**R**	**F**
1	$\exists i,j.\ S_{i,j} = \tau.P_{i,j}$	$\beta[\ ]_\delta$	$*$	$\star$
2	$\exists i,j.\ S_{i,j} = \sigma Z$	$\beta[\ ]_{\{(\tau.X_1+Y_1)/\delta Z\}\circ\delta}$	$*$	
3		$C'[\ ]_\theta\ +\ \tau.X_1$ $C'[\ ]_\theta \mid (\tau.X_1 + Y_1)$ $\tau.C'[\ ]_\theta + Y_1$		
4	$\exists i,j,i',j'.\ i \neq i' \wedge$ $S_{i,j} = r().P_{i,j}\ \wedge\ S_{i',j'} = \overline{r}'s.P_{i',j'}$	$\beta\iota[\ ]_\delta$	$*$	$\star$
5	$\exists i,j.\ S_{i,j} = r().P_{i,j}$	$(\overline{r}'s.X_1 + Y_1) \mid (\sigma[\ ]_\delta\ +\ Y_3)$	$*$	$\star$
6	$\exists i,j.\ S_{i,j} = \overline{r}s.P_{i,j}$	$(r'().X_1 + Y_1) \mid (\sigma[\ ]_\delta\ +\ Y_3)$	$*$	$\star$
7	$\exists i,j,i',j'.\ i \neq i' \wedge$ $S_{i,j} = \sigma_1 Z\ \wedge\ S_{i',j'} = \sigma_2 Z'$	$\beta\iota[\ ]_{\{(r().X_1+Y_1)/\delta Z,(\overline{r}'s.X_3+Y_3)/\delta Z'\}\circ\delta}$	$*$	$\star$
7’	$\exists i,j,i',j'.\ i \neq i' \wedge$ $S_{i,j} = \sigma_1 Z\ \wedge\ S_{i',j'} = \sigma_2 Z'$	$\beta\iota[\ ]_{\{(r().X_1+Y_1)/\delta Z,\nu(\overline{r}'r_0.X_3+Y_3)/\delta Z'\}\circ\delta}$	$*$	
8	$\exists i,j.\ S_{i,j} = \sigma' Z$	$(\overline{r}'s.X_1 + Y_1) \mid \sigma[\ ]_{\{(r().X_3+Y_3)/\delta Z\}\circ\delta}$	$*$	$\star$
9	$\exists i,j.\ S_{i,j} = \sigma' Z$	$(r'().X_1 + Y_1) \mid \sigma[\ ]_{\{(\overline{r}s.X_3+Y_3)/\delta Z\}\circ\delta}$	$*$	$\star$
9’	$\exists i,j.\ S_{i,j} = \sigma' Z$	$(r'().X_1 + Y_1) \mid \sigma[\ ]_{\{\nu(\overline{r}r_0.X_3+Y_3)/\delta Z\}\circ\delta}$	$*$	
10	$\exists i\ m_i = 1\ \wedge\ S_{i,1} = \sigma X$	$\beta[\ ]_{\{((r().X_1+Y_1)\ \mid\ (\overline{r}'s.X_3+Y_3))/\delta X\}\circ\delta}$	$*$ $r \neq r'$	$\star$ $r \neq r'$
10’	$\exists i\ m_i = 1\ \wedge\ S_{i,1} = \sigma X$	$\beta[\ ]_{\{((r().X_1+Y_1)\ \mid\ \nu(\overline{r}'r_0.X_3+Y_3))/\delta X\}\circ\delta}$		
11	$\exists i,j,i',j'.\ i \neq i' \wedge$ $S_{i,j} = \sigma Z\ \wedge\ S_{i',j'} = r().P_{i',j'}$	$\beta\iota[\ ]_{\{(\overline{r}'s.X_1+Y_1)/\delta Z\}\circ\delta}$	$*$	$\star$
11’	$\exists i,j,i',j'.\ i \neq i' \wedge$ $S_{i,j} = \sigma Z\ \wedge\ S_{i',j'} = r().P_{i',j'}$	$\beta\iota[\ ]_{\{\nu(\overline{r}'r_0.X_1+Y_1)/\delta Z\}\circ\delta}$	$*$	
12	$\exists i,j,i',j'.\ i \neq i' \wedge$ $S_{i,j} = \sigma Z\ \wedge\ S_{i',j'} = \overline{r}s.P_{i',j'}$	$\beta\iota[\ ]_{\{(r'().X_1+Y_1)/\delta Z\}\circ\delta}$	$*$	$\star$
13		$C'[\ ]_\theta \mid (r().X_1 + Y_1) \mid (\overline{r}s.X_3 + Y_3)$ $(\overline{r}s.X_1 + Y_1) \mid (C'[\ ]_\theta\ +\ r().X_3)$ $(\overline{r}s.X_1 + Y_1) \mid (r().C'[\ ]_\theta + Y_3)$		

where:

- the substitution $\delta = [X_{2h}/X_h,\ Y_{2h}/Y_h]_{h \geq 0}$ sends all variables into variables with even index;
- $C'[\ ]_\theta$ in rows 3 and 13 is any second-order context s.t. the variables in the GIPO context are not in the codomain of θ;
- r, r' are such that $[\![\mathbb{C} \cdot P, r]\!] = [\![\mathbb{C} \cdot P, r']\!]$;
- if $\mathbb{C}$ is of the form $\beta\iota\mathbb{C}'$, then ι is the identity if $[\![\mathbb{C}' \cdot P, r]\!] = [\![\mathbb{C}' \cdot P, r']\!]$, and a singleton substitution otherwise.

$*$ where β, if it appears, is the identity.

$\star$ where β and σ, if they appear, are the identity.

Table 2. π-calculus final GIPO contexts for closed processes

	Process $P \equiv \nu^k(\Sigma_{j=1}^{m_1} S_{1,j} \mid \ldots \mid \Sigma_{j=1}^{m_n} S_{n,j} \mid \sigma X)$	**GIPO Context** $\mathbb{C}$
1	$\exists i, j.\ S_{i,j} = \tau.P_{i,j}$	$[\ ]_{id}$
2	$\exists i, j, i', j'.\ i \neq i' \ \wedge\ S_{i,j} = r().P_{i,j} \ \wedge\ S_{i',j'} = \overline{r}'s.P_{i',j'}$	$\iota[\ ]_{id}$
3	$\exists i, j.\ S_{i,j} = r().P_{i,j}$	$[\ ]_{\{\overline{r}'s.X_1+Y_1/\delta X\}\circ\delta}$
4	$\exists i, j.\ S_{i,j} = \overline{r}s.P_{i,j}$	$[\ ]_{\{r'().X_1+Y_1/\delta X\}\circ\delta}$
5		$[\ ]_{\{(r().X_1+Y_1 \mid \overline{r'}s.X_3+Y_3)/\delta X\}\circ\delta}$ $r \neq r'$

where:
- r, r' are such that $[\![\mathbb{C} \cdot P, r]\!] = [\![\mathbb{C} \cdot P, r']\!]$;
- if $\mathbb{C}$ is of the form $\iota[\ ]_{id}$, then ι is the identity if $[\![P, r]\!] = [\![P, r']\!]$, and a singleton substitution otherwise.

3 GIPO Bisimilarity on Standard Syntax vs. Open Bisimilarity

In this section, first we provide a presentation of GIPO LTS and bisimilarity for closed processes in the standard π-calculus syntax. Then, we compare this bisimilarity with Sangiorgi's open bisimilarity, [14]. GIPO bisimilarity turns out to be finer than open bisimilarity; however a small variant of it gives exactly the open bisimilarity. Thus, interestingly enough, we obtain an efficient characterization of open bisimilarity, alternative to Sangiorgi's characterization on the symbolic LTS, [14]. An advantage of our presentation lies in the fact that our bisimilarity has a direct definition from the LTS, without requiring the extra machinery of *distinctions*.

3.1 A Presentation of GIPO Bisimilarity on Standard Syntax

In order to compare our GIPO LTS and bisimilarity with standard LTS's and bisimilarities of π-calculus, it is useful to provide a presentation of GIPO LTS and bisimilarity for closed processes in the standard π-calculus syntax.

The intuitive idea is the following. The LTS in Table 2 uses terms having form $\nu^k(P \mid \sigma X)$. In the standard syntax, there is an immediate correspondent for the part $\nu^k(P)$, that is the corresponding nameful π-calculus term. Less obvious is how to define a correspondent for the σX part. The permutation σ essentially depends on output actions that have been performed in the previous transitions (history), and there are three important aspects: (i) the permutation σ is determined by the list of names that have been communicated by the process P to X (the observer); (ii) σ determines which private names in $\nu^k(P)$ can be used for future communications; (iii) through transitions of kind 5 in Table 2, we can check which public name has been communicated to X, and whether the same private name has been used in two different communications. Given the above observations, we represent the information captured by σX via the list L of private names communicated to X by the process. We omit public names, since they can be represented directly on the labels of the LTS, and their presence in the list is not strictly necessary. Thus in the LTS we consider pairs $\langle \nu \boldsymbol{a} Q, L\rangle$ such that the elements of L are names in $\boldsymbol{a}$. Possible applications of the α-rule to the process apply also to the list of names L.

Traditional LTS's use as labels part of the term, dually (G)RPO LTS's use as labels contexts that can interact with the term, and in particular with the part of the term that is "put in evidence" by the traditional LTS; in this presentation we use a traditional approach.

Labels α in the LTS range over $\alpha ::= \tau \mid \{a'/a\} \mid xy \mid \overline{x}y$, where we assume the set of names ordered, and we denote by $\{a'/a\}$ a singleton substitution, with $a < a'$ in such ordering.

Transitions $\langle P, L\rangle \xrightarrow{\alpha} \langle P', L'\rangle$ are described in Table 3.

Table 3. Transitions in the standard LTS

	Process	**List**	**Label**	**Process**	**List**
	$P \equiv \nu \boldsymbol{a}(\Sigma_{j=1}^{m_1} S_{1,j} \mid \ldots \mid \Sigma_{j=1}^{m_n} S_{n,j})$	L	α	P'	L'
1	$\exists i,j.\ S_{i,j} = \tau.P_{i,j}$		τ	$P' \equiv \nu \boldsymbol{a}(\ldots \mid P_{ij} \mid \ldots)$ $L' \equiv L$	
2	$\exists i,j,i',j'.\ (i \neq i' \ \wedge\ S_{i,j} = a(b).P_{i,j}\ \wedge$ $S_{i',j'} = \overline{a}c.P_{i',j'})$		τ	$P' \equiv \nu \boldsymbol{a}(\ldots \mid P_{ij}\{c/b\} \mid \ldots$ $\ldots \mid P_{i'j'} \mid \ldots)$ $L' \equiv L$	
3	$\exists i,j,i',j'.\ (i \neq i' \ \wedge\ S_{i,j} = a(b).P_{i,j}\ \wedge$ $S_{i',j'} = \overline{a}'c.P_{i',j'})$ $a, a' \in \mathit{free}(P),\ a < a'$		$\{a'/a\}$	$P' \equiv (\nu \boldsymbol{a}(\ldots \mid P_{ij}\{c/b\} \mid \ldots$ $\ldots \mid P_{i'j'} \mid \ldots))\{a'/a\}$ $L' \equiv L$	
4	$\exists i,j.\ S_{i,j} = a(b).P_{i,j}\ \wedge\ a \in \mathit{free}(P) \cup L$		xy	$P' \equiv \nu \boldsymbol{a}(\ldots \mid P_{ij}\{c/b\} \mid \ldots)$ $(c \notin \mathit{bn}(P) \vee c \in L)\ \wedge\ L \equiv L'$	
5	$\exists i,j.\ S_{i,j} = \overline{a}c.P_{i,j}\ \wedge\ a \in \mathit{free}(P) \cup L$		$\overline{x}y$	$P' \equiv \nu \boldsymbol{a}(\ldots \mid P_{ij} \mid \ldots)$ $L' \equiv \begin{cases} L & \text{if } c \in \mathit{free}(P) \\ L : c & \text{otherwise} \end{cases}$	

where substitution is capture-avoiding, *i.e.* α-conversion is possibly applied before applying substitution; $x \equiv \begin{cases} a & \text{if } a \in \mathit{free}(P) \\ \nu a & \text{otherwise} \end{cases}$ and $y \equiv \begin{cases} c & \text{if } c \notin \mathit{bn}(P) \\ \nu c & \text{otherwise} \end{cases}$

The resulting LTS is quite similar to symbolic LTS, the main difference being that, for input transitions, in the symbolic LTS just one name is considered, while in the present LTS also previously communicated private names (recorded in the list) are considered.

In order to define the bisimilarity induced by the above LTS, we first need to define a relation on possibly bound names w.r.t. lists of names:

Definition 9. *Let L, M be name lists. We define*
$x =_{LM} y$ *iff* $x = a = y$ *or* $x = \nu a \ \wedge\ y = \nu a' \ \wedge\ \forall i.\ (a = L(i) \iff a' = M(i))$.

The above relation on names can be naturally extended to labels. Then, the GIPO bisimilarity can be recovered on standard π-calculus as the canonical bisimilarity induced by the LTS above, up-to the use of the relation $=_{LM}$ on labels instead of equality. That is, for P, Q processes on the standard syntax, $\emptyset$ the empty list, and $\mathcal{T}(P), \mathcal{T}(Q)$ the translations of P,Q in the syntax with de Brujin indexes, we have:

Theorem 1. $(P, \emptyset) \sim (Q, \emptyset)$ *iff* $\mathcal{T}(P) \sim_C \mathcal{T}(Q)$.

3.2 GIPO Bisimilarity vs. Syntactical and Open Bisimilarity

On the π-fragment without the ν-operator, the list L disappears in our LTS, hence the GIPO bisimilarity coincides with the *syntactical bisimilarity*, which is defined as the bisimilarity induced by the symbolic LTS on the π-fragment without ν [3]. Syntactical bisimilarity is a variant of the open bisimilarity. The latter is defined on the symbolic LTS, by relaxing the condition on fusion transitions: a fusion transition $P \xrightarrow{\{a'/a\}} P'\{a'/a\}$ can be simulated either by a transition with the same fusion label or by a τ-transition $Q \xrightarrow{\tau} Q'$ such that $P'\{a'/a\}$ and $Q'\{a'/a\}$ are bisimilar. The standard definition of open bisimilarity on the full calculus uses the extra machinery of *distinctions*. An interesting result that we obtain is that a small variation of our bisimilarity $\sim$ coincides with the open bisimilarity $\sim_O$ on the full calculus. Namely, let $\approx$ denote the bisimilarity obtained from $\sim$ by allowing a fusion transition to be simulated either by the same fusion or by a τ-transition (where in the resulting process we apply the fusion). Then we have:

Theorem 2. $\approx = \sim_O$.

The above theorem (whose proof is sketched in [12]) gives us a new efficient characterization of the open bisimilarity. The most evident difference between our presentation and the standard presentation is that in the latter distinctions are needed, while we use lists in the LTS but no distinctions. An explanation for this is that, when comparing two terms that can perform an input transition, the open bisimilarity considers just one transition on a free name, while we consider also all the transitions, where a previously communicated bound name (contained in the list L) is received.

Finally, notice that the asymmetric definition of $\approx$ for fusion labels follows the pattern of the *semi-saturated bisimilarity* introduced in [6].

4 Conclusions and Future Work

We have applied the GRPO construction to the full π-calculus, using two extra ingredients. Firstly, we have worked in a category of second-order contexts, based on a presentation of π-calculus with de Bruijn indexes. Secondly, a general pruning technique has been applied, in order to simplify the LTS obtained by the standard (G)RPO construction. Finally, the application of a more ad-hoc simplification technique has allowed us to get an efficient LTS and bisimilarity, and a new characterization of Sangiorgi's open bisimilarity. As it often happens, also in the present case Leifer-Milner technique by itself does not directly give an efficient LTS and bisimilarity. However, this technique, applied in the setting of second-order contexts and in combination with our general pruning technique, gives us substantially less redundant LTS's and bisimilarities, and leads us to the final efficient presentation. Moreover, new insights on the calculus are obtained by applying this machinery. The construction presented in this paper is solid under variations of π-calculus syntax, *e.g.* including replication or match/mismatch operators. In conclusion, the results obtained for π-calculus in this paper and for CCS in [7] are quite promising; in particular, they show that the Leifer-Milner technique is valuable in suggesting interesting notions of LTS's and bisimilarities. Therefore, it would be worth to experiment the above machinery on more recent calculi, for which the notions of LTS and bisimilarity are still evolving.

References

1. 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)
2. Sassone, V., Sobocinski, P.: Deriving bisimulation congruences using 2-categories. Nord. J. Comput. 10, 163–190 (2003)
3. Ferrari, G.L., Montanari, U., Tuosto, E.: Model checking for nominal calculi. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 1–24. Springer, Heidelberg (2005)
4. Gadducci, F., Montanari, U.: Observing reductions in nominal calculi via a graphical encoding of processes. 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. 106–126. Springer, Heidelberg (2005)
5. Bonchi, F., Gadducci, F., König, B.: Process bisimulation via a graphical encoding. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 168–183. Springer, Heidelberg (2006)
6. Bonchi, F., König, B., Montanari, U.: Saturated semantics for reactive systems. In: LICS, pp. 69–80. IEEE Computer Society, Los Alamitos (2006)
7. Di Gianantonio, P., Honsell, F., Lenisa, M.: Finitely branching labelled transition systems from reaction semantics for process calculi. In: WADT. LNCS, vol. 5486, pp. 119–134. Springer, Heidelberg (2009)
8. Bonchi, F., Gadducci, F., Monreale, G.V.: Reactive systems, barbed semantics, and the mobile ambients. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 272–287. Springer, Heidelberg (2009)
9. Di Gianantonio, P., Honsell, F., Lenisa, M.: RPO, second-order contexts, and lambda-calculus. Logical Methods in Computer Science 5 (2009)
10. Jensen, O.H., Milner, R.: Bigraphs and transitions. In: POPL, pp. 38–49 (2003)
11. Sobocinski, P.: A well-behaved lts for the pi-calculus (abstract). Electr. Notes Theor. Comput. Sci. 192, 5–11 (2007)
12. Di Gianantonio, P., Jaksic, S., Lenisa, M.: Efficient bisimilarities from second-order reaction semantics for pi-calculus. Technical report, Università di Udine (2010), http://sole.dimi.uniud.it/~marina.lenisa/Papers/Soft-copy-pdf/tr10.pdf
13. Parrow, J.: An introduction to the pi-calculus. In: Bergstra, Ponse, Smolka (eds.) Handbook of Process Algebra, pp. 479–543. Elsevier, Amsterdam (2001)
14. Sangiorgi, D.: A theory of bisimulation for the pi-calculus. Acta Inf. 33, 69–97 (1996)

On the Use of Non-deterministic Automata for Presburger Arithmetic

Antoine Durand-Gasselin and Peter Habermehl

LIAFA, CNRS & University Paris Diderot (Paris 7), Case 7014, 75205 Paris 13, France
{adg,haberm}@liafa.jussieu.fr

Abstract. A well-known decision procedure for Presburger arithmetic uses deterministic finite-state automata. While the complexity of the decision procedure for Presburger arithmetic based on quantifier elimination is known (roughly, there is a double-exponential non-deterministic time lower bound and a triple exponential deterministic time upper bound), the exact complexity of the automata-based procedure was unknown. We show in this paper that it is triple-exponential as well by analysing the structure of the non-deterministic automata obtained during the construction. Furthermore, we analyse the sizes of deterministic and non-deterministic automata built for several subclasses of Presburger arithmetic such as disjunctions and conjunctions of atomic formulas. To retain a canonical representation which is one of the strengths of the use of automata we use residual finite-state automata, a subclass of non-deterministic automata.

1 Introduction

Presburger arithmetic (PA) is the first-order theory over integers with addition. It has many applications for example in system verification, constraint data bases, etc. PA was shown decidable [13] using the method of quantifier elimination. The complexity of the decision problem has been well studied. In [9] a double exponential non-deterministic time lower bound was given and Oppen [8] showed that Cooper's [5] quantifier elimination algorithm has triple exponential worst-case complexity in deterministic time. The bound was improved [14] considering the number of quantifier eliminations.

Another decision procedure for PA is based on the use of deterministic finite-state word automata (DFA). The idea comes from Büchi [4]. Integer vectors are represented as words over a suitable alphabet and an automaton for a given formula accepts exactly the words which correspond to vectors making the formula true. An automaton representing exactly the solutions of a formula can be constructed recursively from a formula by using automata constructions corresponding to the logical connectives and quantifiers. Eliminating an existential quantifier is done by projecting away the track corresponding to the given variable and then determinising and minimising the resulting automaton. An advantage of the approach is that a canonical representation — the minimal automaton — is obtained for a formula. This allows simple equivalence checking of two formulas which is often needed in verification applications. The automata based approach has been studied for example by [3,16] who gave direct constructions of automata for atomic linear constraints. It is implemented for example in the tools LASH

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 373–387, 2010.

[17] and FAST [1,18]. A rigorous analysis of the size of the minimal DFA for a Presburger formula is given in [10]. The tight upper bound obtained is triple-exponential in the size of the formula. The result is shown not by analysing the automata based decision procedure outlined above but by eliminating the quantifiers, thus obtaining a quantifier-free formula and analysing the size of the automaton constructed from it. The exact size of the intermediate automata in the automata based decision procedure was left open. More precisely, there might have been an exponential blow-up coming from the determinisation, leading to automata with a quadruple exponential size.

In this paper we investigate the use of *non-deterministic automata* (NFA) for PA which has not yet been extensively studied. To retain the property of canonicity we study a subclass of NFA called residual finite-state automata (RFSA) [6] which admit a canonical minimal automaton. States of RFSA accept residuals of the language accepted by the automaton. Our two main contributions are: (1) Though Klaedtke [10] has shown using results of [9] that the lower bound for the size of the NFA for a Presburger formula is the same as the triple-exponential lower bound for the size of the DFA, it is interesting to investigate the potential gain of the use of NFA. We show first that for atomic formulas no gain at all can be achieved. Then, we show that for several subclasses of Presburger formulas (for example for disjunctions of inequations) a substantial gain in the size of the automata is obtained. For this we characterise exactly the number of states of the minimal DFA and compare them with the minimal RFSA. The results obtained are also of interest for constructing tools for Presburger arithmetic, since we show for some subclasses that the automata obtained by the usual product construction are minimal. (2) We show that the size of the automata obtained during the automata based decision procedure is triple-exponential solving an open problem from [10]. This is done by carefully inspecting the structure of the NFA obtained after projection. We show that the determinisation of this automaton does not lead to an exponential blow-up. From that follows a triple-exponential time upper bound for the automata based decision procedure.

2 Preliminaries

Presburger Arithmetic is the first-order theory on atomic formulas of the form $\sum_{i=1}^{n} a_i x_i \sim c$, where a_i and c are integer constants, x_i integer variables, and $\sim$ is an operator among $\{=, \neq, <, > \leq, \geq, \equiv_m\}$ (with $\equiv_m$ the congruence modulo m, for any $m \geq 2$).

We will restrict ourselves here to the case where $\sim \in \{=, >, \equiv_m\}$ and $gcd(a_1, \ldots, a_n) = 1$, and also consider the two trivial formulas $\top$ and $\bot$ (respectively the tautology and the unsatisfiable proposition) as atomic formulas. As boolean operators we use $\neg$, $\wedge$ and $\vee$.

A Presburger formula is defined as an object of this theory. The length of a Presburger formula is defined in the usual way (see [10]). We use the vectorial notation for the atomic formulas, i.e. $\mathbf{a}.\mathbf{x} \sim c$. Let φ be a Presburger formula with k free variables. $[\![\varphi]\!]$ is defined as the set of solutions of φ, that is the set of assignments (seen as a subset of $\mathbb{Z}^k$) of the free variables of φ validating φ (with the usual semantics of arithmetic). The set of solutions (a set of integer vectors) of any Presburger formula is called a Presburger set. Two Presburger formulas φ and φ' are called semantically equivalent iff $[\![\varphi]\!] = [\![\varphi']\!]$. Two n-tuples of Presburger formulas $(\varphi_1, \ldots, \varphi_n)$ and $(\varphi'_1, \ldots, \varphi'_n)$ are called semantically equivalent iff $[\![\varphi_i]\!] = [\![\varphi'_i]\!]$ for all $i \in \{1, \ldots, n\}$. For $1 \leq k \leq n$, we let $\mathbf{e}^n_k$ be the n-dimensional vector which has a 1 as its k-th component and 0 elsewhere.

Finite Word Automata. Let Σ be a finite alphabet and Σ^* the set of finite words over Σ. The empty word is denoted by ϵ and the length of a word u by $|u|$. A *non-deterministic finite automaton* (NFA) $\mathcal{A}$ is a 5-tuple $\langle \Sigma, Q, Q_0, F, \delta \rangle$, with Σ a finite alphabet, Q a finite set of states, $Q_0 \subseteq Q$ (resp. $F \subseteq Q$) the initial (resp. final) states and δ the transition function mapping $Q \times \Sigma$ to 2^Q. An NFA $\mathcal{A} = \langle \Sigma, Q, Q_0, F, \delta \rangle$ is said to be deterministic (DFA) if Q_0 is a singleton $\{q_0\}$ and if $\forall q \in Q,\ \forall x \in \Sigma,\ Card(\delta(q, x)) \leq 1$. Therefore the transition function δ of a DFA can be seen as a partial application of $Q \times \Sigma$ to Q.

A path labelled by $u = u_1 \cdots u_n$ in an NFA $\mathcal{A} = \langle \Sigma, Q, Q_0, F, \delta \rangle$ from $q_0 \in Q$, is a sequence of states $q_0, q_1, \ldots, q_n$ such that $\forall i, 1 \leq i \leq n, q_i \in \delta(q_{i-1}, u_i)$. We define inductively $\widehat{\delta} : 2^Q \times \Sigma^* \to 2^Q$ as $\widehat{\delta}(Q', \epsilon) = Q'$ and $\widehat{\delta}(Q', au) = \widehat{\delta}(\bigcup_{q' \in Q'} \delta(q', a), u)$, that is the set of states that can be reached by a path labelled by u from a state of Q'. A path $q_0, \ldots, q_n$ in $\mathcal{A}$ is accepting if $q_n \in F$. A word u is accepted by $\mathcal{A}$ if there is a path $q_0, \ldots, q_n$ labelled by u in $\mathcal{A}$ such that $q_0 \in Q_0$ and $q_n \in F$. The language $L_{\mathcal{A}}$ is the set of words accepted by $\mathcal{A}$. The reverse of u, noted $\widetilde{u}$ is inductively defined as $\widetilde{\epsilon} = \epsilon$ and $\widetilde{x \cdot v} = \widetilde{v} \cdot x$, for $x \in \Sigma$ and $v \in \Sigma^*$. The reverse language $\widetilde{L}$ of $L \subseteq \Sigma^*$ is defined as $\widetilde{L} = \{\widetilde{u} \in \Sigma^* \mid u \in L\}$. The reverse automaton of an NFA $\mathcal{A} = \langle \Sigma, Q, Q_0, F, \delta \rangle$ is defined as $\widetilde{\mathcal{A}} = \langle \Sigma, Q, F, Q_0, \widetilde{\delta} \rangle$ with $q \in \widetilde{\delta}(q', x)$ if and only if $q' \in \delta(q, x)$. Clearly, $\widetilde{L_{\mathcal{A}}} = L_{\widetilde{\mathcal{A}}}$.

We define the notion of residual languages (or simply *residuals*) both for automata and languages. Let L be a language over an alphabet Σ and w a word of Σ^*. The residual language or left quotient of L by w, is $w^{-1}L = \{u \mid w \cdot u \in L\}$. Given an automaton $\mathcal{A}$, a residual (left-residual) of a state q, called $L_{\mathcal{A},q}$ or $\mathrm{post}_{\mathcal{A},q}$, is defined as $L_{\mathcal{A},q} = \{u \mid F \cap \widehat{\delta}(\{q\}, u) \neq \emptyset\}$. A right-residual of a state q, is defined as $\mathrm{pre}_{\mathcal{A},q} = \{u \mid q \in \widehat{\delta}(Q_0, u)\}$.

Obviously, residuals of states of a deterministic automaton are always residuals of the language that the automaton accepts (provided that all states are reachable).

The *size* of an automaton is defined the usual way, i.e. the number of states plus the number of transitions. An automaton is said to be minimal in a class of automata if any other automaton in it accepting the same language has at least as many states. It is well known that minimal DFA are canonical (unique up to isomorphism) whereas NFA do not have a minimal canonical form. Thus, we use RFSA [6], a subclass of NFA having a minimal canonical form. The number of states of the minimal RFSA lies between the number of states of the minimal NFA and the number of states of the minimal DFA.

Let $\mathcal{A}$ be an NFA, $\mathcal{A} = \langle \Sigma, Q, Q_0, F, \delta \rangle$. $\mathcal{A}$ is a *residual finite state automaton* (RFSA) if for all state $q \in Q$, $L_{\mathcal{A},q}$ is a residual language of $L_{\mathcal{A}}$. Formally, $\forall q \in Q$, $\exists u \in \Sigma^*$, such that $L_{\mathcal{A},q} = u^{-1}L_{\mathcal{A}}$. Therefore any DFA where all states are reachable, is an RFSA, but the converse is not true, and for some languages (like $(a+b)^*a(a+b)^n$, see [6]), the smallest RFSA is exponentially smaller than the minimal DFA.

Let L be a regular language over Σ, and u a word of Σ^*. The residual language $u^{-1}L$ is said to be *prime*, if it is not equal to the union of the residual languages of L it strictly contains. Let $R_u = \{v \in \Sigma^* | v^{-1}L \subsetneq u^{-1}L\}$, $u^{-1}L$ is a prime residual of L if $\bigcup_{v \in R_u} v^{-1}L \subsetneq u^{-1}L$. Intuitively, a residual language of L is prime if it cannot be obtained as a union of other residual languages of L. Each regular language L is accepted by a minimal canonical RFSA [6] defined below. Each of its states is a prime residual of L.

Definition 1. *Let L be a regular language over Σ. Its canonical RFSA $(\Sigma, Q, Q_0, F, \delta)$ is defined as follows. Q is the set of prime residuals of L, i.e. $Q = \{u^{-1}L \mid u^{-1}L$ is prime$\}$. Initial states are prime residual languages contained in L, i.e. $Q_0 = \{u^{-1}L \in Q \mid u^{-1}L \subseteq L\}$ and final states are prime residual languages containing the empty word, i.e. $F = \{u^{-1}L \in Q \mid \epsilon \in u^{-1}L\}$. The transition function is defined by $\delta(u^{-1}L, a) = \{v^{-1}L \in Q \mid v^{-1}L \subseteq (ua)^{-1}L\}$, for $u^{-1}L \in Q$ and $a \in \Sigma$.*

Encoding of Presburger sets as languages. We represent integer vectors as finite words, so we can map Presburger sets to languages. We use a vectorial least significant bit first coding. For $k > 0$ we define $\Sigma_k = \{0, 1\}^k$. Moreover we use the separate sign alphabet $S_k = \{+, -\}^k$ (indicating if the corresponding integer is positive or negative). Words of $\Sigma_k^* S_k$ represent k-dimensional integer vectors. A word $w_0 \dots w_n s \in \Sigma_k^* S_k$ represents the integer vector we denote $\langle w_0 \dots w_n s\rangle$, each component of which is computed as follows (where $\pi_i(x)$ representing the value of the i-th component of x). If $s_i = +$, then $\pi_i(\langle w_0 \dots w_n s\rangle) = \sum_{j=0}^{n} 2^j.\pi_i(w_j)$ and if $s_i = -$, then $\pi_i(\langle w_0 \dots w_n s\rangle) = -2^{n+1} + \sum_{j=0}^{n} 2^j.\pi_i(w_j)$. In particular, $\langle +\rangle = 0$ and $\langle -\rangle = -1$. We also define the notation $\langle .\rangle_+$ over Σ_k^* as $\langle w\rangle_+ = \langle w+^k\rangle$.

Remark 1. Let $w', w \in \Sigma_k^*$, $s \in S_k$. We have $\langle w'ws\rangle = \langle w'\rangle_+ + 2^{|w'|}\langle ws\rangle$.

This representation is clearly surjective and provides an infinite number of representations for each vector. Indeed for any word $w_0 \dots w_n s \in \Sigma_k^* S_k$, any $w_0 \dots w_n (s')^* s$ (with $s'_i = 0$ if $s_i = +$ or $s'_i = 1$ if $s_i = -$) represents the same vector.

Given a Presburger formula $\varphi(\mathbf{x})$ (with $\mathbf{x}$ the vector of free variables of φ, and k its dimension), we say it defines the language $L_\varphi = \{w \in \Sigma_k^* S_k \mid \langle w\rangle \in [\![\varphi]\!]\}$. Such languages are called Presburger-definable and meet the following saturation property: if a representation of a vector is in the language then any other representation of that vector is also in the language. Our coding meets the following important property [12].

Property 1. Any residual of a saturated Presburger-definable language is either a saturated Presburger-definable language, or the empty word language.

Thus we not only have a residual closure property on Presburger-definable sets, but we also have an effective way to characterise them. In an automaton accepting all solutions of a Presburger formula $\varphi(\mathbf{x})$ with k free variables, a word $w \in \Sigma_k^*$ leads from the initial state to a state accepting exactly all solutions of $\varphi(2^{|w|}\mathbf{x} + \langle w\rangle_+)$.

Automata representing atomic Presburger formula. We study here automata that accept Presburger-definable languages. Notice that all final states of such automata are equivalent (there is only one residual that contains the empty word). All other states have a residual that is Presburger definable, i.e. definable by a Presburger formula. Thus, in the following, automata have as states a unique final state F and elements of $\mathcal{F}$, the set of Presburger formulas. Each state $\varphi \in \mathcal{F}$ characterises its residual, for example a state with empty residual will be represented by $\perp$.

We recall here the construction of the DFA for atomic Presburger constraints. The construction for equations and inequations with a least significant bit first coding was given by Boudet and Comon [3]; as they worked with natural numbers, they had not to handle the sign letters. Wolper and Boigelot [16] worked with integers but used a most significant bit first coding. They gave a backwards construction of the automaton yielding an NFA for inequations which they then had to determinise. Since we work with a least significant bit first coding we are spared this determinisation procedure. We also give the construction of the automaton for modular constraints directly, rather than using Klaedtke's [10] which is also based on a most significant bit first coding.

- The automaton $A_{\mathbf{a}.\mathbf{x}=c}$ for the formula $\mathbf{a}.\mathbf{x} = c$ is given by the following forward construction starting from the initial state $\mathbf{a}.\mathbf{x} = c$:
  - if $b \in \Sigma_k$, $\delta(\mathbf{a}.\mathbf{x} = \gamma, b) = \begin{cases} \mathbf{a}.\mathbf{x} = \gamma' \text{ with } \gamma' = \frac{\gamma - \mathbf{a}.\mathbf{b}}{2} \text{ when } 2 \mid \gamma - \mathbf{a}.\mathbf{b} \\ \perp \text{ otherwise} \end{cases}$
  - if $s \in S_k$, $\delta(\mathbf{a}.\mathbf{x} = \gamma, s) = F$ when $\mathbf{a}.\langle s \rangle = \gamma$ and $\delta(\mathbf{a}.\mathbf{x} = \gamma, s) = \perp$ otherwise
  - for $\alpha \in S_k \cup \Sigma_k, \delta(\perp, \alpha) = \delta(F, \alpha) = \perp$
- The automaton $A_{\mathbf{a}.\mathbf{x}>c}$ for the formula $\mathbf{a}.\mathbf{x} > c$ is given by the following forward construction starting from the initial state $\mathbf{a}.\mathbf{x} > c$:
  - if $b \in \Sigma_k$, $\delta(\mathbf{a}.\mathbf{x} > \gamma, b) = \mathbf{a}.\mathbf{x} > \gamma'$ with $\gamma' = \left\lfloor \frac{\gamma - \mathbf{a}.\mathbf{b}}{2} \right\rfloor$.
  - if $s \in S_k$, $\delta(\mathbf{a}.\mathbf{x} > \gamma, s) = F$ when $\mathbf{a}.\langle s \rangle > \gamma$ and $\delta(\mathbf{a}.\mathbf{x} > \gamma, s) = \perp$ otherwise
  - for $\alpha \in S_k \cup \Sigma_k, \delta(\perp, \alpha) = \delta(F, \alpha) = \perp$
- The automaton $A_{\mathbf{a}.\mathbf{x} \equiv_{2^m(2n+1)} c}$ for the formula $\mathbf{a}.\mathbf{x} \equiv_{2^m(2n+1)} c$ (with some $m, n \geq 0$) is given by the following forward construction starting from $\mathbf{a}.\mathbf{x} \equiv_{2^m(2n+1)} c$:
  - if $b \in \Sigma_k$ and $p \geq 1$,
    $\delta(\mathbf{a}.\mathbf{x} \equiv_{2^p(2n+1)} \gamma, b) = \begin{cases} \mathbf{a}.\mathbf{x} \equiv_{2^{p-1}(2n+1)} \gamma' \text{ with } \gamma' = \frac{\gamma - \mathbf{a}.\mathbf{b}}{2} \text{ when } 2 \mid \gamma - \mathbf{a}.\mathbf{b} \\ \perp \text{ otherwise} \end{cases}$
  - if $b \in \Sigma_k$ and $p = 0$,
    $\delta(\mathbf{a}.\mathbf{x} \equiv_{2n+1} \gamma, b) = \mathbf{a}.\mathbf{x} \equiv_{2n+1} \gamma'$ with $\gamma' = \begin{cases} \frac{\gamma - \mathbf{a}.\mathbf{b}}{2} \text{ when } 2 \mid \gamma - \mathbf{a}.\mathbf{b} \\ \frac{\gamma + 2n + 1 - \mathbf{a}.\mathbf{b}}{2} \text{ when } 2 \nmid \gamma - \mathbf{a}.\mathbf{b} \end{cases}$
  - if $s \in S_k$, $\delta(\mathbf{a}.\mathbf{x} \equiv_{2^p(2n+1)} \gamma, s) = \begin{cases} F \text{ when } \mathbf{a}.\langle s \rangle \equiv_{2^p(2n+1)} \gamma \\ \perp \text{ otherwise} \end{cases}$
  - for $\alpha \in S_k \cup \Sigma_k, \delta(\perp, \alpha) = \delta(F, \alpha) = \perp$

The correctness of this construction derives from Property 1. For all states φ and all $b \in \Sigma_k$ we have $\delta(\varphi(\mathbf{x})) = \varphi(2\mathbf{x} + b)$ showing that the transitions labelled by Σ_k are correct. The transitions labelled by S_k are correct by definition, as the last letter is always from S_k. The definitions are clearly exhaustive. Furthermore, as $gcd(\mathbf{a}) = 1$, all states are non-equivalent, indeed $[\![\mathbf{a}.\mathbf{x} \sim \gamma]\!] \neq [\![\mathbf{a}.\mathbf{x} \sim \gamma']\!]$ when $\gamma \neq \gamma'$. Thus our construction provides us with the *minimal DFA*. In the following we detail the number of states of the automata constructed. Those results were given by Klaedtke [10] on automata for most significant bit first coding. They are easily adapted for equations and inequations. For a vector $\mathbf{a}$ we define $||\mathbf{a}||_+ = \sum_{\{i \mid a_i \geq 0\}} a_i$ and $||\mathbf{a}||_- = \sum_{\{i \mid a_i \leq 0\}} |a_i|$.

Theorem 1. *Let* $gcd(\mathbf{a}) = 1$. *For the case of an equation* $\mathbf{a}.\mathbf{x} = c$*, the states* $\{\mathbf{a}.\mathbf{x} = \gamma \mid -||a||_+ < \gamma < ||a||_-\}$ *are reachable and form a maximal strongly connected component (SCC) and all other reachable states are in* $\{\mathbf{a}.\mathbf{x} = \gamma \mid \gamma = c \vee \min(c, -||a||_+) < \gamma < \max(c, ||a||_-)\}$. *In the case of an inequation* $\mathbf{a}.\mathbf{x} > c$*, the states* $\{\mathbf{a}.\mathbf{x} > \gamma \mid -||\mathbf{a}||_+ \leq \gamma < ||\mathbf{a}||_-\}$ *are reachable and form a maximal SCC and all other reachable states are in* $\{\mathbf{a}.\mathbf{x} > \gamma \mid \gamma = c \vee \min(c, -||a||_+) \leq \gamma < \max(c, ||a||_-)\}$. *For modulo constraints*

$\mathbf{a}.\mathbf{x} \equiv_{2^n(2p+1)} c$ *the states in* $\{\mathbf{a}.\mathbf{x} \equiv_{2p+1} \gamma \mid \gamma \in [0, 2p]\}$ *are reachable and form a maximal SCC and all other reachable states are in* $\{\mathbf{a}.\mathbf{x} \equiv_{2^m(2p+1)} \gamma \mid (m = n \wedge \gamma = c) \vee (m < n \wedge \gamma \in [0, 2^m(2p+1) - 1])\}$.

The SCCs only depend on $\mathbf{a}$ and not on the constant c. Before proving the theorem we give in Figure 1 the automaton for a simple inequation. The reachable maximal SCC is formed by the states $\{-1, 0, 1, 2\}$. The transitory state 3 exists only because $3 > ||\mathbf{a}||_-$.

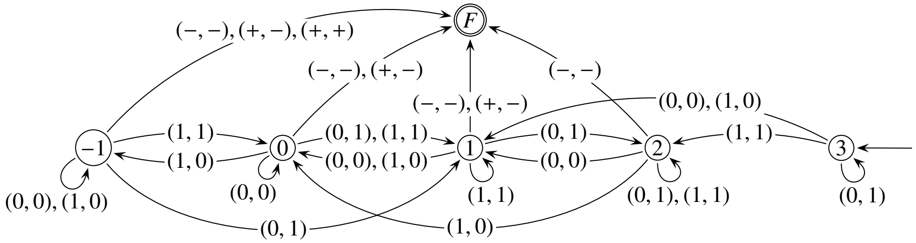


Fig. 1. An automaton for $x - 3y > 3$

Proof. Klaedtke [10] studied the number of reachable states for equations and inequations. Automata for equations are by construction backwards deterministic, thus we built the reverse automaton of Klaedtke's and all states in $\{\mathbf{a}.\mathbf{x} = \gamma \mid -||a||_+ < \gamma < ||a||_-\}$ are reachable and form a maximal SCC. We get the same reachable states forming a maximal SCC for an automaton for an inequation $\mathbf{a}.\mathbf{x} > \gamma$, as it just contains more transitions than the automaton for $\mathbf{a}.\mathbf{x} = \gamma$. For modulo constraints, for our coding, if the modulus is even, it is halved by the next transition, so all states are reached only for odd modulus. Thus, the automaton is possibly smaller compared to the automaton constructed for the most significant bit first coding which is as big as the modulus. □

3 Gain of Non-determinism

We study here the potential gain of using NFA, in particular RFSA, for subclasses of Presburger arithmetic. We start with the following negative result for atomic constraints.

Theorem 2. *For equations, inequations or modulo constraints with an odd modulus, no NFA is smaller than the minimal DFA.*

The proof of this theorem consists essentially in considering a class of languages for which the minimal DFA is already a minimal NFA. We use the following result [11] for biRFSA languages (i.e. languages accepted by biRFSAs). An automaton $\mathcal{A}$ is a biRFSA if $\mathcal{A}$ and $\widetilde{\mathcal{A}}$ are both RFSA.

Proposition 1. *The canonical RFSA of a biRFSA language is a minimal NFA.*

We will show that the languages we consider are biRFSA and that their canonical RFSA is not smaller than the minimal DFA. Therefore it is the minimal NFA as well.

Lemma 1. *Given a language L with residuals $L_0 = L, L_1, \dots, L_n$, such that: (1) Given a residual, residuals languages included in it form a strictly increasing chain w.r.t. inclusion, i.e. $\forall L_i, L_j, L_k . L_j \subseteq L_i \wedge L_k \subseteq L_i \implies L_j \subseteq L_k \vee L_k \subseteq L_j$, (2) Two residuals are either disjoint or one is included in the other, i.e. $\forall L_i, L_j . L_i \cap L_j \neq \emptyset \implies L_i \subseteq L_j \vee L_j \subseteq L_i$. Then any NFA accepting this language is at least as big (in term of number of states) as the minimal DFA.*

Proof. Let L be a regular language satisfying the lemma's hypotheses. The minimal DFA has as many states as the canonical RFSA as all the residuals are prime. This is because for $L_0, \dots, L_m$ residuals of L, if $L_0 = \bigcup_{i \geq 1} L_i$ then $\forall i . L_i \in L_0$ thus the L_i form an increasing chain, thus $\exists i \geq 1 . L_0 = L_i$.

Let $\mathcal{A} = \langle \Sigma, (q_i)_{i \leq m}, Q_0, F, \delta \rangle$ be the canonical RFSA of L. We show that L is biRFSA by observing that the reversed automaton of the canonical RFSA is a RFSA. With the two hypotheses we have on the residuals, it is easy to find for each residual L_i (suppose of the state q_i) a word w_i that is only in L_i and the residuals containing L_i.

Let us show that $\mathrm{post}_{\widetilde{\mathcal{A}}, q_i} = \widetilde{w_i}^{-1} \widetilde{L}$. By definition $\widetilde{w_i}^{-1} \widetilde{L} = \{ v \mid \widetilde{w_i} . v \in \widetilde{L} \}$, then $v \in \mathrm{post}_{\widetilde{\mathcal{A}}, q_i} \Rightarrow \widetilde{v} \in \mathrm{pre}_{\mathcal{A}, q_i} \Rightarrow \widetilde{v} . w_i \in L \Rightarrow \widetilde{w_i} . v \in \widetilde{L}$. We have the first inclusion, $\mathrm{post}_{\widetilde{\mathcal{A}}, q_i} \subseteq \widetilde{w_i}^{-1} \widetilde{L}$ and now show the other inclusion, $\widetilde{w_i} . v \in \widetilde{L} \Rightarrow v \in \mathrm{post}_{\widetilde{\mathcal{A}}, q_i}$. If $\widetilde{w_i} . v \in \widetilde{L}$, then by definition, there is an accepting path labelled by $\widetilde{w_i} . v$ in $\widetilde{\mathcal{A}}$. We can extract from this path a path labelled by $\widetilde{w_i}$, which will reach a state, namely q. By definition of w_i, $\mathrm{post}_{\mathcal{A}, q} \supseteq \mathrm{post}_{\mathcal{A}, q_i}$, from which we deduce that $\mathrm{pre}_{\mathcal{A}, q} \subseteq \mathrm{pre}_{\mathcal{A}, q_i}$, from which we deduce that $\mathrm{post}_{\widetilde{\mathcal{A}}, q} \subseteq \mathrm{post}_{\widetilde{\mathcal{A}}, q_i}$. We have shown that the $\mathrm{post}_{\widetilde{\mathcal{A}}, q_i}$ are residuals of $\widetilde{L}$. Thus $\widetilde{\mathcal{A}}$ is a RFSA, so $\mathcal{A}$ is a biRFSA, and L a biRFSA language. There is no smaller NFA than the canonical RFSA which has as many states as the minimal DFA. □

Now, Theorem 2 follows by observing that the residuals of the corresponding languages verify the conditions of Lemma 1.

Boolean combinations of atomic constraints. We first consider general boolean combinations of atomic constraints. A boolean combination of formulas $\varphi_1, \dots, \varphi_n$ is a formula generated by $\top, \bot, \varphi_1, \dots, \varphi_n, \neg, \vee$ or $\wedge$. We denote by $C(\varphi_1, \dots, \varphi_n)$ such a boolean combination. We define the notion of *simple* boolean combination as follows. The underlying propositional formula corresponding to $C(\varphi_1, \dots, \varphi_n)$ is $C(b_1, \dots, b_n)$ where $b_1, \dots, b_n$ are propositional variables. We say that $C(b_1, \dots, b_n)$ is simple, if the truth value of the formula depends on all propositional variables. Then, a boolean combination $C(\varphi_1, \dots, \varphi_n)$ is simple, if its underlying propositional formula $C(b_1, \dots, b_n)$ is simple. From any boolean combination $C(\varphi_1, \dots, \varphi_n)$ we can always obtain a simple one by removing some atomic formulas if needed.

To build an automaton for a boolean combination of atomic formulas, we build a product automaton whose states are Presburger formulas (not tuples of formulas).

Definition 2. *Given a boolean combination of atomic formulas $C(\varphi_1(\mathbf{x}), \dots, \varphi_n(\mathbf{x}))$, the product automaton $A_{C(\varphi_1(\mathbf{x}), \dots, \varphi_n(\mathbf{x}))}$ is given by: Q is the set of Presburger formulas and the designated final state F, $q_0 = C(\varphi_1(\mathbf{x}), \dots, \varphi_n(\mathbf{x}))$ and for all $b \in \Sigma_k$, $\delta(C(\psi_1(\mathbf{x}), \dots, \psi_n(\mathbf{x})), b) = C(\psi'_1(\mathbf{x}), \dots, \psi'_n(\mathbf{x}))$ each $\psi_i(\mathbf{x})$ being a state, possibly $\bot$, of $\mathcal{A}_{\varphi_i}$ (the automaton of φ_i), and $\psi'_i(\mathbf{x}) = \delta_{\varphi_i}(\psi_i(\mathbf{x}), b)$. If $s \in S^k$, then $\delta(C(\psi_1(\mathbf{x}), \dots, \psi_n(\mathbf{x})), s) = F$, when $\langle s \rangle \in [\![C(\psi_1(\mathbf{x}), \dots, \psi_n(\mathbf{x}))]\!]$ and $\delta(C(\psi_1(\mathbf{x}), \dots, \psi_n(\mathbf{x})), s) = \bot$ otherwise.*

It is clear that this construction provides us with a deterministic finite automaton. To exhibit the gain of the canonical RFSA over the minimal DFA for the fragment of boolean combinations of inequations, we need to characterise precisely the number of states of the minimal DFA and the canonical RFSA.

Proposition 2. *If we consider a simple boolean combination of inequations $C(\mathbf{a}_1.\mathbf{x} > c_1, \ldots, \mathbf{a}_n.\mathbf{x} > c_n)$, such that $(\mathbf{a}_i)_{1 \le i \le n}$ form a linearly independent family of vectors, then the product automaton is the minimal DFA.*

Proof. To prove this proposition we need first a simple lemma from linear algebra.

Lemma 2. *If $(\mathbf{a}_i)_{1 \le i \le n}$ is a linearly independent family of vectors, then there is a family of integer vectors $(\mathbf{u}_1, \ldots, \mathbf{u}_n)$ such that $\mathbf{a}_i.\mathbf{u}_j = 0$ for all j with $i \neq j$ and $\mathbf{a}_i.\mathbf{u}_i \neq 0$.*

We show that any two states of the automaton for the formula $C(\mathbf{a}_1.\mathbf{x} > c_1, \ldots, \mathbf{a}_n.\mathbf{x} > c_n)$ are non-equivalent by exhibiting a word that is in one's residual and not in the other's, i.e. a representation of a solution of the formula characteristic of one state that is not a solution of the formula characteristic of the other. We consider two states $C(\mathbf{a}_1.\mathbf{x} > c_1, \ldots, \mathbf{a}_n.\mathbf{x} > c_n)$ and $C(\mathbf{a}_1.\mathbf{x} > c'_1, \ldots, \mathbf{a}_n.\mathbf{x} > c'_n)$. They are distinct if there is an i with $c_i \neq c'_i$ (we assume w.l.o.g that $c_i < c'_i$). We can find an integer vector $\mathbf{v}$ with $\mathbf{a}_i.\mathbf{v} = c'_i$. As the boolean combination is simple we can find $b_1, \ldots, b_n$ booleans such that (seeing C as a boolean function) $C(b_1, \ldots, b_{i-1}, true, b_{i+1}, \ldots, b_n) \neq C(b_1, \ldots, b_{i-1}, false, b_{i+1}, \ldots, b_n)$. For each $j \neq i$, we can find an integer λ_j with $\mathbf{a}_j.(\mathbf{v} + \lambda \mathbf{u}_j) > max(c_j, c'_j)$ if b_j is true or $\mathbf{a}_j.(\mathbf{v} + \lambda_j \mathbf{u}_j) < min(c_j, c'_j)$ if b_j is false (the u_j's are taken from the family defined in Lemma 2). By definition $\mathbf{a}_j.(\mathbf{v} + \sum_{k \neq i} \lambda_k \mathbf{u}_k) = \mathbf{a}_j.(\mathbf{v} + \lambda_j \mathbf{u}_j)$, thus $(\mathbf{v} + \sum_{k \neq i} \lambda_k \mathbf{u}_k)$ is a solution of only one of the two states: they are not equivalent. As no two states are equivalent, the automaton is minimal. □

We now analyse the number of states of the automaton for disjunctions of atomic inequations $\mathbf{a}.\mathbf{x} > c$. We have shown for one inequation that the set of states can be partitioned into two subsets: a set (depending on $\mathbf{a}$) which is a max. SCC and a possibly empty set of other states that reach this SCC. The product automaton has the same structure. Notice that we can canonically map the max. SCC of the automaton for an atomic inequation to an interval. As we use a forward construction of the automaton, not every state in the cartesian product of the basic automata is reached. We now determine precisely the shape of the maximal SCC in the product automaton.

Given a boolean combination of inequations $C(\mathbf{a}_1.\mathbf{x} > c_1, \ldots, \mathbf{a}_n.\mathbf{x} > c_n)$, we will characterise the set of states (aside from $\perp$ and F) of the product automaton we reach by the forward construction. It should be clear that the set of states will be a subset of $\{C(\mathbf{a}_1.\mathbf{x} > \gamma_1, \ldots, \mathbf{a}_n.\mathbf{x} > \gamma_n) \mid \gamma_i \in \mathbb{Z}\}$. We will therefore map states to elements of $\mathbb{Z}^n$. We define the polyhedron $\Pi = \{\mathbf{v} \in \mathbb{Q}^n \mid \exists \lambda \in]0, 1[^n, \mathbf{v} = (-\mathbf{a}_1.\lambda, \ldots, -\mathbf{a}_n.\lambda)\}$.

Theorem 3. *Given a boolean combination of inequations, the set of integer points in $\mathcal{P} = \Pi +]-1, 0[^n$ form a maximal SCC in the corresponding product automaton.*

Proof. We show in two steps that every integer point in $\mathcal{P}$ is reachable from any state. First we compute a candidate word, then we show that this candidate allows us to reach that state. Suppose we want to reach a state $\gamma \in \Pi +]-1, 0[^n$ from the state $\mathbf{c}$.

What makes this reachability problem difficult is that we work with integers. Let us first define a similar reachability problem in $\mathbb{Q}^n$. Let $w = b_1 \dots b_p$ be a word of Σ_k. We define inductively the sequence $(\mathbf{r}_j)_{0 \le j \le p}$ as $\mathbf{r}_0 = \mathbf{c}$ and $\mathbf{r}_{j+1} = \frac{\mathbf{r}_j - (\mathbf{a}_1.b_{j+1}, \cdots, \mathbf{a}_n.b_{j+1})}{2}$. We can deduce that $\mathbf{r}_p = \frac{\mathbf{c}}{2^p} - \frac{(\mathbf{a}_1.\langle w \rangle_+, \cdots, \mathbf{a}_n.\langle w \rangle_+)}{2^p}$.

As $\gamma \in \Pi +]-1, 0[^n$, we can write $\gamma = \mathbf{u} + \mathbf{e}$ with $\mathbf{u} \in \Pi$ and $\mathbf{e} \in]-1, 0[^n$. By definition of Π, there exists $\lambda \in]0, 1[^n$ such that $\mathbf{u} = -(\mathbf{a}_1.\lambda, \dots, \mathbf{a}_n.\lambda)$. It should be clear that we can choose w such that $\frac{\langle w \rangle_+}{2^{|w|}}$ is arbitrarily close to any vector of $[0, 1]^n$, thus arbitrarily close to λ, and therefore we can choose w such that $\frac{(\mathbf{a}_1.\langle w \rangle_+, \dots, \mathbf{a}_n.\langle w \rangle_+)}{2^p}$ is arbitrarily close to $\mathbf{u}$. If we prefix w by $(0, \dots, 0)^*$ we do not change the value of $\frac{\langle w \rangle_+}{2^{|w|}}$ but we increase its length, and thus $\frac{\mathbf{c}}{2^{|w|}}$ gets arbitrarily small. Therefore we can choose w such that $\mathbf{r}_p$ is arbitrarily close to $\mathbf{u}$.

We now have a candidate w to reach the state γ: we want to show that the path labelled by w in the product automaton reaches γ. We define the sequence $(\mathbf{q}_j)_{0 \le j \le p}$ that represents the path of w in the product automaton: $\mathbf{q}_0 = \mathbf{c}$ and $\mathbf{q}_{j+1} = \left\lfloor \frac{\mathbf{q}_j - (\mathbf{a}_1.b_{j+1}, \dots, \mathbf{a}_n.b_{j+1})}{2} \right\rfloor$ (where $\lfloor \cdot \rfloor$ means applying the floor function componentwise), $\mathbf{q}_p$ is the state reached by w from $\mathbf{c}$. We can show inductively that $0 \le \mathbf{r}_j - \mathbf{q}_j < 1$, indeed $\mathbf{r}_{j+1} - \mathbf{q}_{j+1} = \frac{\mathbf{r}_j - \mathbf{q}_j}{2} + \left(\frac{\mathbf{q}_j - (\mathbf{a}_1.b_{j+1}, \dots, \mathbf{a}_n.b_{j+1})}{2} - \left\lfloor \frac{\mathbf{q}_j - (\mathbf{a}_1.b_{j+1}, \dots, \mathbf{a}_n.b_{j+1})}{2} \right\rfloor \right)$, so we can deduce that $\mathbf{q}_p = \lfloor \mathbf{r}_p \rfloor$.

By definition of $\lfloor . \rfloor$, there is $\mathbf{f} \in [0, 1[^n$ such that $\lfloor \mathbf{r}_p \rfloor = \mathbf{r_p} + \mathbf{f}$, thus $\mathbf{q}_p = \gamma + \mathbf{f} - (\mathbf{e} + (\mathbf{u} - \mathbf{r}_p))$. As $\mathbf{r}_p$ is arbitrarily close to $\mathbf{u}$, $\mathbf{e} + (\mathbf{u} - \mathbf{r}_p) \in] - 1, 0[^n$. Since $\mathbf{q}_p$ and γ have integer components and $\mathbf{f} \in [0, 1[^n$, we have $\mathbf{q}_p = \gamma$.

Thus, any integer point in $\mathcal{P}$ is a state of the product automaton, $\mathcal{P}$ forms an SCC which is maximal, as from any state $\gamma \in \mathcal{P}$ one can only reach states inside $\mathcal{P}$. □

Figure 2 gives an example of the states of the product automaton reached for a disjunction of two simple inequations together with its polyhedra Π and $\mathcal{P}$.

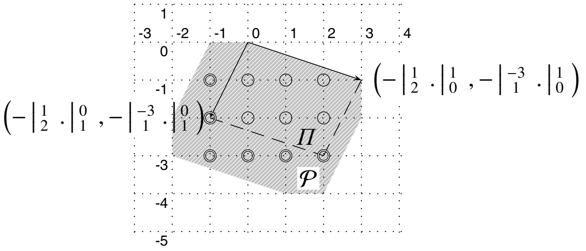

1
-3
-2
-1
0
1
2
3
4
0
-1
-3
-4
-5
$\left(-\begin{vmatrix}1\\2\end{vmatrix} \cdot \begin{vmatrix}1\\0\end{vmatrix}, -\begin{vmatrix}-3\\1\end{vmatrix} \cdot \begin{vmatrix}1\\0\end{vmatrix}\right)$
$\left(-\begin{vmatrix}1\\2\end{vmatrix} \cdot \begin{vmatrix}0\\1\end{vmatrix}, -\begin{vmatrix}-3\\1\end{vmatrix} \cdot \begin{vmatrix}0\\1\end{vmatrix}\right)$
Π
$\mathcal{P}$

Fig. 2. $\mathbb{Z}^2$-states for $x - 3y > c_1 \vee 2x + y > c_2$, the doubly circled states are the states of the canonical RFSA

We precisely characterised the set of states forming the maximal SCC constructed by our forward construction of the automaton for disjunctions of inequations. This automaton is always minimal when the inequations are linearly independent.

We study now what the gain of using non-determinism is. We analyse the number of states of RFSA for conjunctions and disjunctions of inequations. The case of disjunction of inequations appears to be the most simple for building non-deterministic automata. For example we can take the simple union of the automata for each inequation of the disjunction. However they are not necessarily RFSA.

Proposition 3. *Given a disjunction of inequations, let Q be the set of states of the minimal DFA seen as a subset of $\mathbb{Z}^n$, by canonically mapping $\bigvee_{i=1}^n \mathbf{a}_i.\mathbf{x} > \gamma_i$ to $(\gamma_1, \dots, \gamma_n)$. Then, the states of the canonical RFSA are in $\mathcal{S} = \{\gamma \in Q \mid \{\gamma\} + \{-1, 0\}^n \not\subset Q\}$.*

Proof. By definition, the residual of a state γ is not prime if and only if there exists a family of states $(\gamma^{(k)})$ $(\gamma \notin (\gamma^{(k)}))$, such that the residual of $\bigvee_{i=1}^{n} \mathbf{a}_i.\mathbf{x} > \gamma_i$ is equivalent to the union over k of the residuals of $\bigvee_{i=1}^{n} \mathbf{a}_i.\mathbf{x} > \gamma_i^{(k)}$. If $\gamma \notin \mathcal{S}$, the $(\gamma^{(k)} = \gamma - \mathbf{e}_k^n)$ are all states in Q, it is straightforward to notice that $\bigvee_i \mathbf{a}_i.\mathbf{x} > \gamma_i = \bigcup_i \bigvee_j \mathbf{a}_j.\mathbf{x} > \gamma_j^{(i)}$, thus any state not in $\mathcal{S}$ is not prime. □

We can then heuristically characterise the gain in the number of states of the RFSA in comparison with the minimal DFA. Instead of the whole polyhedron representing the max. SCC, the RFSA only has at most the states of its lower surface $\mathcal{S}$.

The RFSA is in general bigger than the non-deterministic automaton built as union of automata. Disjunctions of inequations having not many variables in common have small minimal RFSA compared to minimal DFA, for example the minimal DFA of a formula of the form $\bigvee_{i=1}^{n/2} x_{2i} + x_{2i+1} > 0$ has an exponential number of states in n, whereas the minimal RFSA has a linear number of states.

Actually even for a fixed number of variables we can have an exponential gain. For the language $\cup_{l \leq n} \Sigma_1^l.1.\Sigma_1^n.1.0^*.+$ (which only represents a finite, thus Presburger, set of integers) the minimal RFSA is exponentially smaller than the minimal DFA. It also shows that there are Presburger sets for which the most significant bit first coding is accepted by an exponentially smaller automaton compared to the least significant bit first coding.

We now provide a characterisation of the canonical RFSA for a conjunction of inequations, which surprisingly can be as small as the canonical RFSA for disjunctions.

Proposition 4. *Given a conjunction of inequations $\bigwedge_i \mathbf{a}_i.\mathbf{x} > c_i$, let Q be the set of states of its minimal DFA, seen as a subset of $\mathbb{Z}^n$. Let $\mathcal{S}' = \{\gamma \in Q \mid \exists k, \gamma + \mathbf{e}_k^n \notin Q\}$ and $\mathcal{G} = \{\gamma \mid \bigwedge_{i=1}^{n} \mathbf{a}_i.\mathbf{x} = \gamma_i + 1$ has no integer solution$\}$. The set of states (seen as elements of $\mathbb{Z}^n$) of the canonical RFSA for this conjunction is a subset of $\mathcal{S}' \cup \mathcal{G}$. If $\mathbf{c}$, the initial state, is in $\mathcal{P}$ and the $\mathbf{a}_i$ are linearly independent, this is exactly the set of states of the canonical RFSA.*

Proof. Here again, the overapproximation is proved by looking at the definition of an RFSA. We remark that $\bigvee_k \bigwedge_i \mathbf{a}_i.\mathbf{x} > \gamma_i + \delta_k(i)$ ($\delta_k(i) = 1$ when $k = i$ and 0 otherwise) is equivalent to $\bigwedge_i \mathbf{a}_i.\mathbf{x} > \gamma_i \land \neg \bigvee_i \gamma_i + 1 \geq \mathbf{a}_i.\mathbf{x} > \gamma_i$. If $\bigwedge_{i=1}^{n} \mathbf{a}_i.\mathbf{x} = \gamma_i + 1$ has no integer solution, this is equivalent to $\bigwedge_i \mathbf{a}_i.\mathbf{x} > \gamma_i$, thus if a state is not in $\mathcal{S}' \cup \mathcal{G}$, it is not prime.

We now prove that we have characterised the states with prime residuals when the $\mathbf{a}_i$ are linearly independent and the initial state is in $\mathcal{P}$. The second hypothesis allows us to use the convexity of $\mathcal{P}$. We assume that the union of the residuals of the states of the family $(\gamma^{(k)})$ $(\gamma \notin (\gamma^{(k)}))$ is equal to the residual of γ. Then, we show that $\gamma \in \mathcal{S}' \cup \mathcal{G}$. As the $\mathbf{a}_i$ are linearly independent, for any j and κ, $\bigwedge_{i \neq j} \mathbf{a}_i.\mathbf{x} = \gamma_i + 1 \land \mathbf{a}_j.\mathbf{x} > \kappa$ characterises an affine half-space of dimension at least 1. Thus such a system has infinitely many solutions. So, for any k, only the residuals of the states $\bigwedge_{i \neq j} \mathbf{a}_i.\mathbf{x} > \gamma_i' + 1 \land \mathbf{a}_j.\mathbf{x} > \gamma_j'$ with $\gamma_i' \leq \gamma_i$ (for all $i \neq j$) contain these solutions. Only those states with $\gamma_i' \geq \gamma_i$ are included in the residual of γ. Thus $(\gamma^{(k)})$ necessarily contains, for all j a $(\gamma_1, \ldots, \gamma_{j-1}, \kappa_j, \gamma_{j+1}, \ldots)$ with some $\kappa_j > \gamma_j$. With the convexity of $\mathcal{P}$, we have $\gamma \notin \mathcal{S}'$. □

4 Complexity of the Automata Based Decision Procedure

The well-known decision procedure for Presburger arithmetic using automata is based on recursively constructing an automaton accepting solutions of a Presburger formula by using automata constructions for handling logical connectives and quantifiers. Automata for basic formulas can be easily constructed (see section 2). Each logical connective ($\wedge$,$\vee$, $\neg$) corresponds then naturally to an operation on automata. Furthermore to get an automaton for $\exists y.\varphi(y, \mathbf{x})$ given an automaton for $\varphi(y, \mathbf{x})$ one *projects away* [1] the component for y and obtains a *non-deterministic* automaton. Then, this automaton is determinised to be able to continue the recursive construction and minimised. Given a formula of size n, Klaedtke has shown [10] — by eliminating the quantifiers on the logical level and translating the obtained quantifier-free formula to an automaton — that the minimal DFA obtained *at the end* of this procedure and the minimal DFA for subformulas obtained *during* the procedure have triple-exponential size. However, the DFA obtained after determinising the NFA which is a result of a projection might be of quadruple exponential size, as an automaton of triple exponential size is determinised. In this section we show that *all* automata obtained during the construction are in fact of at most triple exponential size solving an open problem from [10]. We do this by carefully inspecting the structure of the NFA which is determinised. Notice that our upper bound is obtained using the least significant bit first coding of integer vectors which allows to reason conveniently about states of the automata corresponding to formulas.

The following theorem on quantifier elimination is from Klaedtke [10]. We use here a simplified version, where the only parameter is the length of the formula. In [10], other parameters (e.g. alternation depth) are used. Given a quantifier free Presburger formula ψ, let d_ψ be the number of different atomic modulo constraints of the form $\mathbf{a}.\mathbf{x} \equiv_m \beta$ and $max_{div}(\psi)$ the biggest value of m appearing in them. Let t_ψ be the number of *different* vectors $\mathbf{a}$ appearing in atomic formulas of the form $\mathbf{a}.\mathbf{x} \sim \gamma$ with $\sim \in \{=, \neq, <, >, \leq, \geq, \}$ in ψ, $max_{coef}(\psi)$ the biggest absolute value of their coefficients and $max_{const}(\psi)$ the biggest absolute value of the constants γ appearing in them. We use the abbreviations $exp2(x) = 2^{2^x}$ and $exp3(x) = 2^{2^{2^x}}$.

Theorem 4 ([10], Theorem 4.5). *For every Presburger formula φ of length n, there is a semantically equivalent quantifier free formula ψ such that: $t_\psi \leq exp2(cn \log n)$, $d_\psi \leq exp2(cn \log n)$, $max_{coef}(\psi) < exp2(cn)$, $max_{div}(\psi) < exp2(cn)$ and $max_{const}(\psi) < exp3(cn \log n)$, where c is a constant independent of n.*

We can suppose w.l.o.g. that ψ is a boolean combination of modulo constraints of the form $\mathbf{a}.\mathbf{x} \equiv_m \beta$ and of atomic formulas of the form $\mathbf{a}.\mathbf{x} > \gamma$ only. The following theorem gives a bound on the size of the minimal DFA accepting solutions of a Presburger formula. Klaedtke gives a corresponding theorem for the most significant bit first coding. His proof is simpler due to the fact that only one automaton has to be constructed for all inequations with the same coefficients.

Theorem 5. *The size of the minimal DFA accepting solutions of a Presburger formula φ of length n is at most $exp3(cn \log n)$ for some constant c.*

[1] Since the automaton should accept all encodings of the solutions, one has to sometimes add additional transitions with a sign letter going to the final state.

Proof. Let k be the number of free variables of φ. Let ψ be the quantifier free formula obtained from φ using Theorem 4. We have $t_\psi \leq exp2(c_1 n \log n)$, $d_\psi \leq exp2(c_1 n \log n)$, $max_{coef}(\psi) < exp2(c_1 n)$, $max_{div}(\psi) < exp2(c_1 n)$ and $max_{const}(\psi) < exp3(c_1 n \log n)$ for some constant c_1. If we build the product automaton for the quantifier free formula ψ equivalent to φ according to Definition 2, a naive analysis of its size gives a quadruply exponential automaton, since there are possibly $t_\psi^{2max_{const}(\psi)}$ distinct inequations in ψ. However a closer inspection reveals a triple exponential bound, due to the special structure of automata for inequations. Here, we give a slightly different construction of the automaton A_ψ accepting solutions of ψ which we will use in the rest of the section.

Let $\mathbf{a}_1, \ldots, \mathbf{a}_{t_\psi}$ be the different vectors appearing in the atomic inequations of ψ and $\psi_1, \ldots, \psi_{l_\psi}$ an enumeration of all atomic formulas of the form $\mathbf{a}_i.\mathbf{x} > \gamma_j$ for all $1 \leq i \leq t_\psi$ and γ_j with $|\gamma_j| \in [-||\mathbf{a}_i||_+ - 1, ||\mathbf{a}_i||_-]$. Clearly, $l_\psi \leq exp2(c_2 n \log n)$ for some constant c_2. Let $\phi_1, \ldots, \phi_{d_\psi}$ be an enumeration of all the modulo constraints appearing in ψ and $\mathcal{BC}$ be the set of boolean combinations of the form $C(\psi_1, \ldots, \psi_{l_\psi}, \phi_1, \ldots, \phi_{d_\psi})$. For each member of $\mathcal{BC}$ an automaton can be built with the product construction of Definition 2.

We describe now informally the automaton A_ψ we construct from ψ. It has first the form of a complete tree starting at the initial state. Its branching factor is the size of the alphabet Σ_k and its depth is $exp2(c_1 n \log n)$. Each of the states in the tree recognises the solutions of the formula $\psi(2^{|w|}\mathbf{x} + \langle w \rangle_+)$ where $w \in \Sigma_k^*$ with $|w| \leq exp2(c_1 n \log n)$ is the word leading to the state from the initial state. Then, at level $exp2(c_1 n \log n)$ there are separate automata accepting solutions of the corresponding formulas reached after reading the word leading to them. All these automata correspond to boolean combinations of $\mathcal{BC}$. Indeed, for any atomic formula $\zeta(\mathbf{x}) = \mathbf{a}.\mathbf{x} > \gamma$ of ψ and any word $w \in \Sigma_k^*$ with $|w| = exp2(c_1 n \log n)$ we have $\zeta(2^{|w|}\mathbf{x} + \langle w \rangle_+) \Leftrightarrow \mathbf{a}.\mathbf{x} > \gamma'$ for some $\gamma' \in [-||a||_+ - 1, ||a||_-]$. Therefore, for any atomic subformula $\zeta(\mathbf{x})$ of ψ, $\zeta(2^{|w|}\mathbf{x} + \langle w \rangle_+)$ is equivalent to a ψ_i, so $\psi(2^{|w|}\mathbf{x} + \langle w \rangle_+)$ is equivalent to a formula of $\mathcal{BC}$. Notice that in any member of $\mathcal{BC}$ *all* atomic formulas of a given form appear. That is not a restriction, since we can just expand each boolean combination to be of this form. Let $W = \{w \in \Sigma_k^* \mid |w| = exp2(c_1 n \log n)\}$. For any $w \in W$, let $C_w \in \mathcal{BC}$ be the boolean combination equivalent to $\psi(2^{|w|}\mathbf{x} + \langle w \rangle_+)$. For each C_w we can construct an automaton $A_{C_w} = (Q_w \cup \{F\}, q_{w,0}, \{F\}, \delta_w)$ according to Definition 2. Notice that the automata A_{C_w} only differ in the transitions going to the final state, since the atomic formulas composing them are all the same. The final state F is the same in each automaton.

We can now give the definition of the automaton for the formula ψ formally, i.e. $A_\psi = (Q, q_\epsilon, \{F\}, \delta)$ where $Q = Q_1 \cup Q_2 \cup \{F\}$ with $Q_1 = \{q_w \mid w \in \Sigma_k^* \wedge |w| < exp2(c_1 n \log n)\}$ and $Q_2 = \bigcup_{w \in W} Q_w$. Furthermore, $\delta(q_w, b) = \{q_{wb}\}$ for all $b \in \Sigma_k$ and $|w| < exp2(c_1 n \log n) - 1$, $\delta(q_w, b) = \{q_{wb,0}\}$ for all $b \in \Sigma_k$ and $|w| = exp2(c_1 n \log n) - 1$ and $\delta(q, b) = \delta_w(q, b)$ for all $b \in \Sigma_k$ and $q \in Q_2$. Clearly, the number of states (and also the size) of the automaton A_ψ is smaller than $exp3(cn \log n)$ for some constant c. □

We can now prove the main theorem of the section which shows that elimination of a variable does not lead to an exponential blow-up in the size of the automaton.

Theorem 6. *Let $\exists y.\varphi(y, \mathbf{x})$ be a Presburger formula of size n, A the minimal DFA accepting the solutions of $\varphi(y, \mathbf{x})$ and A' the automaton obtained by projecting A on $\mathbf{x}$. Then, the automaton A'' obtained by determinising A' with the standard on-the-fly subset construction is of size at most $exp3(cn \log n)$ for some constant c.*

Proof. Theorem 4 yields a quantifier free formula ψ semantically equivalent to $\varphi(y, \mathbf{x})$ with $t_\psi \leq exp2(c_1 n \log n)$, $d_\psi \leq exp2(c_1 n \log n)$, $max_{coef}(\psi) < exp2(c_1 n)$, $max_{div}(\psi) < exp2(c_1 n)$ and $max_{const}(\psi) < exp3(c_1 n \log n)$ for some constant c_1. For ψ, according to Theorem 5, there is an automaton $A_\psi = (Q, q_0, \{F\}, \delta)$ of triple-exponential size (not necessarily minimal) accepting the solutions of ψ. We use the same notation as in the proof of Theorem 5 for the parts of the automaton. A is the minimal automaton corresponding to A_ψ. Obviously, A_ψ might be bigger than A. We show that the size of the automaton A''_ψ obtained by determinising (using the standard on-the-fly subset construction) A'_ψ which is the automaton obtained from A_ψ by projecting away y is bounded by $exp3(cn \log n)$. Then the bound on A'' (whose states are sets of states of A') follows, as two different states in A'' correspond to two different states in A''_ψ.

Since we use for the determinisation of A'_ψ the standard subset construction, states of A''_ψ are sets of states of A'_ψ which are exactly the states of A_ψ.

We first introduce some notations. Let k be the number of free variables of $\varphi(y, \mathbf{x})$ and ψ. For any word $w \in \Sigma_k^*$ we denote by $w{\downarrow_1} \in \Sigma_{k-1}^*$ the word obtained from w by projecting away the first component and by $w{\downarrow_2} \in \{0, 1\}^*$ the word obtained from w by projecting on the first component. For any $w \in \Sigma_{k-1}^*$ we define $w{\uparrow} = \{w' \in \Sigma_k^* \mid w'{\downarrow_1} = w\}$. For any $w \in \Sigma_{k-1}^*$ and $z \in [0, 2^{|w|} - 1]$ we define $w{\uparrow^z} = w'$, if $\langle w'{\downarrow_2}\rangle_+ = z$ and $w'{\downarrow_1} = w$.

Let $S = \{\widehat{\delta}(w{\uparrow}, \{q_0\}) \mid w \in \Sigma_{k-1}^*\}$. Our goal is to show that the size of S is bounded by a triple-exponential. This implies that the number of states of A''_ψ has the same bound. We split S into two sets $S_<$ and $S_\geq$ where $S_< = \{\widehat{\delta}(w{\uparrow}, \{q_0\}) \mid w \in \Sigma_{k-1}^* \wedge |w| < exp2(c_1 n \log n)\}$ and $S_\geq = \{\widehat{\delta}(w{\uparrow}, \{q_0\}) \mid w \in \Sigma_{k-1}^* \wedge |w| \geq exp2(c_1 n \log n)\}$. It is obvious that $|S_<| \leq exp3(c_2 n \log n)$ for some constant c_2. We now show a bound on $|S_\geq|$. We first enumerate all words $w \in \Sigma_{k-1}^*$ of size exactly $exp2(c_1 n \log n)$ as $w_1, \ldots, w_m$ where $m \leq exp3(c_3 n \log n)$ for some constant c_3. We have $S_\geq = \bigcup_{i=1}^m S_i$ where $S_i = \{\{\widehat{\delta}(w_i w{\uparrow}, \{q_0\}) \mid w \in \Sigma_{k-1}^*\}$. We will show that $|S_i| \leq exp3(c_4 n \log n)$ for some constant c_4 which implies that $S_\geq$ is bounded by a triple-exponential as well. We have $S_i = \{\widehat{\delta}(w{\uparrow}, \widehat{\delta}(w_i{\uparrow}, \{q_0\})) \mid w \in \Sigma_{k-1}^*\} = \bigcup_{z \in [0, (exp3(c_1 n \log n) - 1)]} \{\widehat{\delta}(w{\uparrow}, \widehat{\delta}(w_i{\uparrow^z}, \{q_0\})) \mid w \in \Sigma_{k-1}^*\}$. Let $S_i^0 = \{\widehat{\delta}(w{\uparrow}, \widehat{\delta}(w_i{\uparrow^0}, \{q_0\})) \mid w \in \Sigma_{k-1}^*\}$. We have $|S_i| = |S_i^0|$, as $\widehat{\delta}(w_i{\uparrow^0}, \{q_0\})) = C_{w_i\uparrow^0}$ and for all $0 < z \leq exp3(c_1 n \log n) - 1$ we have $\widehat{\delta}(w_i{\uparrow^z}, \{q_0\}) = C_{w_i\uparrow^z}$ and all automata corresponding to $C_{w_i\uparrow^z}$ for $0 \leq z \leq exp3(c_1 n \log n) - 1$ are the same except for the transitions leading to the final state. That means that there is a one-to-one correspondence between each state in sets of states of S_i^0 and each state in sets of states of the other S_i^z.

Now, we derive a bound of $|S_i^0|$. The word $w_i{\uparrow^0}$ leads to the state $C_{w_i\uparrow^0}$ in A_ψ which is the initial state of an automaton, call it A_0, recognising all solutions of $C_{w_i\uparrow^0}$. A_0 is obtained as a product of automata for atomic inequations and modulo constraints. Let $\psi_1, \ldots, \psi_{l_\psi}$ be the atomic formulas which are inequations and $\phi_1, \ldots, \phi_{d_\psi}$ the atomic formulas which are modulo constraints of the boolean combination $C_{w_i\uparrow^0}$. In the following it is convenient to consider states of A_0 to be $(l_\psi + d_\psi)$-tuples of states instead of considering them as formulas. That is, a state $C(\psi'_1, \ldots, \psi'_{l_\psi}, \phi'_1, \ldots, \phi'_{d_\psi})$ is considered to be the tuple $(\psi'_1, \ldots, \psi'_{l_\psi}, \phi'_1, \ldots, \phi'_{d_\psi})$. For $1 \leq i \leq l_\psi$ let $\psi_i(y, \mathbf{x}) = a_1^i y + \mathbf{a}^i.\mathbf{x} > \gamma_i$ and for $1 \leq i \leq d_\psi$ let $\phi_i(y, \mathbf{x}) = b_1^i y + \mathbf{b}^i.\mathbf{x} \equiv_{m_i} \beta_i$.

Let us fix a $w \in \Sigma_{k-1}^*$ and let $w' \in w{\uparrow}$. Let $y' = \langle w'{\downarrow_2}\rangle_+$ and $l = |w| = |w'|$. It is clear that $0 \leq y' < 2^l$. For each $1 \leq i \leq t_\psi$, the state reached in $A_{a_1^i y + \mathbf{a}^i.\mathbf{x} > \gamma_i}$ by w' is the state semantically equivalent to $a_1^i(2^l y + y') + \mathbf{a}^i.(2^l \mathbf{x} + \langle w \rangle_+) > \gamma_i$ which is equivalent to $a_1^i y + \mathbf{a}^i.\mathbf{x} > (\gamma_i - a_1^i y' - \mathbf{a}.\langle w \rangle_+)/2^l$.

Therefore, the first t_ψ components of the states reached in A_0 by words $w' \in w\uparrow$ are semantically equivalent to $(a_1^1 y + \mathbf{a}^1.\mathbf{x} > (\gamma_1 - a_1^1 y' - \mathbf{a}.\langle w \rangle_+)/2^l, \dots, a_1^{t_\psi} y + \mathbf{a}^{t_\psi}.\mathbf{x} > (\gamma_i - a_1^{t_\psi} y' - \mathbf{a}^{t_\psi}.\langle w \rangle_+)/2^l)$. There are 2^l different values for y'. However there are at most $\sum_{i=1}^{t_\psi}(|a_1^i| + 1)$ semantically different corresponding t_ψ-tuples of formulas, since $0 \leq y' < 2^l$ and therefore the semantics of the i-th atomic formula changes at most a_1^i times in a monotone fashion for increasing y'. Therefore if we consider the first t_ψ components of states reached by words of $w\uparrow$ in A_0, we get only $\sum_{i=1}^{t_\psi}(|a_1^i| + 1)$ semantically different ones, since the automata for basic formulas are minimal.

Now we consider the set of words $V = \{w' \in \Sigma_k^* \mid w'\downarrow_1 = w\}$ which lead to the *same* first t_ψ components of states in A_0 and consider the other components (corresponding to the modulo constraints) they can reach. The words in V differ only in the component corresponding to y. Clearly, the set $\{y' \mid y' = \langle w'\downarrow_2 \rangle_+$ and $w' \in V\}$ is an interval of the form $[p, q]$ where $0 \leq p \leq q < 2^l$.

A state (formula) reached in $A_{b_1^i y + \mathbf{b}^i.\mathbf{x} \equiv_{m_i} \beta_i}$ after reading a word w' of V with $y' = \langle w'\downarrow_2 \rangle_+$ is semantically equivalent to $2^l(b_1^i y + \mathbf{b}^i.\mathbf{x}) \equiv_{m_i} \beta_i - b_1^i y' - \mathbf{b}^i.\langle w' \rangle_+$. It is clear that there are at most m_i semantically different formulas of this kind. Furthermore, we can order them starting from $y' = 0$ until $y' = m_i - 1$. Then it is clear that the set of states (formulas) reached by words of V (whose corresponding y' form intervals) must be an interval of states respecting this order. There are at most m_i^2 such intervals.

Finally, we can give an upper bound on the number of subsets of states of $S_i^0 = \{\widehat{\delta}(w\uparrow, \{q_{w_i,0}\} \mid w \in \Sigma_{k-1}^*\}$ which are subsets of states of A_0. Given any word $w \in \Sigma_{k-1}^*$ we know from the above that words of $w\uparrow$ lead to at most $s := \sum_{i=1}^{t_\psi}(|a_1^i| + 1) \leq exp2(c_5 n \log n)$ (for some constant c_5) different tuples of the first t_ψ components of A_0. Furthermore, we know that the number of subsets of states of A_{ϕ_i} which can be reached simultaneously by words of subsets V of $w\uparrow$ such that all $w' \in V$ lead to the same tuples of the first t_ψ components is at most m_i^2. Therefore overall, S_i^0 has at most $|A_0|^s \Pi_{i=1}^{d_\psi} m_i^2 \leq exp3(cn \log n)$ states for some constant c and $|A_0|$ being the number of states of A_0. From this follows in turn a triple-exponential bound on $|S_i|$, $|S_\geq|$, the number of states and size of A''_ψ and A''. □

The number of transitions of an automaton is bounded by $|Q||\Sigma|$ for a DFA and possibly $|Q|^2|\Sigma|$ for an NFA. As Σ is at most simply exponential w.r.t. the size of the formula, the sizes of the automata build have all a triple-exponential upper bound as well. Therefore the following is an easy consequence of Theorem 6.

Corollary 1. *The automata based decision procedure for Presburger arithmetic takes triple-exponential time in the size of the formula.*

5 Conclusion

We have investigated the use of non-deterministic automata for Presburger arithmetic (PA). We show that for some subclasses NFA might lead to a substantial gain in the size of the automata. We plan to continue our investigations into other subclasses like conjunctions and disjunctions of modulo constraints together with inequations and equations as well as general formulas in conjunctive or disjunctive normal form. The use of alternating automata might improve the sizes as well. But we then lose the canonicity property of RFSA. We also plan to investigate how the use of RFSA improves the performance of tools for PA. RFSA have also been used recently in learning [7,2]. It remains to be seen if these learning algorithms lead to improved performances for learning based verification of counter systems as studied for example in [15] using DFA.

References

1. Bardin, S., Leroux, J., Point, G.: FAST Extended Release. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 63–66. Springer, Heidelberg (2006)
2. Bollig, B., Habermehl, P., Kern, C., Leucker, M.: Angluin-Style Learning of NFA. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), Pasadena, CA, USA, July 2009, pp. 1004–1009. AAAI Press, Menlo Park (2009)
3. Boudet, A., Comon, H.: Diophantine Equations, Presburger Arithmetic and Finite Automata. In: Kirchner, H. (ed.) CAAP 1996. LNCS, vol. 1059, pp. 30–43. Springer, Heidelberg (1996)
4. Büchi, J.: Weak second-order Arithmetic and Finite Automata. Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik 6, 66–92 (1960)
5. Cooper, D.C.: Theorem Proving in Arithmetic without Multiplication. In: Machine Intelligence, pp. 91–100 (1972)
6. Denis, F., Lemay, A., Terlutte, A.: Residual Finite State Automata. Fundamenta Informaticae 51(4), 339–368 (2002)
7. Denis, F., Lemay, A., Terlutte, A.: Learning Regular Languages using RFSAs. Theoretical Comput. Sci. 313(2), 267–294 (2004)
8. Oppen, D.C.: A $2^{2^{2^{pn}}}$ Upper Bound on the Complexity of Presburger Arithmetic. J. Comput. Syst. Sci. 16(3), 323–332 (1978)
9. Fischer, M.J., Rabin, M.O.: Super-exponential Complexity of Presburger Aithmetic. In: Symp. on Applied Mathematics. SIAM-AMS Proceedings, vol. VII, pp. 27–41 (1974)
10. Klaedtke, F.: Bounds on the Automata Size for Presburger Arithmetic. ACM Trans. Comput. Logic 9(2), 1–34 (2008)
11. Latteux, M., Lemay, A., Roos, Y., Terlutte, A.: Identification of biRFSA Languages. Theoretical Computer Science 356(1-2), 212–223 (2006)
12. Leroux, J.: Structural Presburger Digit Vector Automata. Theoretical Computer Science 409(3), 549–556 (2008)
13. Presburger, M.: Uber die Vollstandigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: Comptes Rendus du Premier Congres des Mathematicienes des Pays Slaves, pp. 92–101 (1929)
14. Reddy, C.R., Loveland, D.W.: Presburger Arithmetic with Bounded Quantifier Alternation. In: ACM Symposium on Theory of Computing, pp. 320–325 (1978)
15. Vardhan, A., Viswanathan, M.: Learning to verify branching time properties. Formal Methods in System Design 31(1), 35–61 (2007)
16. Wolper, P., Boigelot, B.: An Automata-theoretic Approach to Presburger Arithmetic Constraints. In: Mycroft, A. (ed.) SAS 1995. LNCS, vol. 983, pp. 21–32. Springer, Heidelberg (1995)
17. Lash homepage, http://www.montefiore.ulg.ac.be/~boigelot/research/lash/
18. FAST homepage, http://www.lsv.ens-cachan.fr/Software/fast/

Reasoning about Optimistic Concurrency Using a Program Logic for History

Ming Fu[1], Yong Li[1], Xinyu Feng[1,2], Zhong Shao[3], and Yu Zhang[1]

[1] University of Science and Technology of China
[2] Toyota Technological Institute at Chicago
[3] Yale University

Abstract. Optimistic concurrency algorithms provide good performance for parallel programs but they are extremely hard to reason about. Program logics such as concurrent separation logic and rely-guarantee reasoning can be used to verify these algorithms, but they make heavy uses of history variables which may obscure the high-level intuition underlying the design of these algorithms. In this paper, we propose a novel program logic that uses invariants on history traces to reason about optimistic concurrency algorithms. We use past tense temporal operators in our assertions to specify execution histories. Our logic supports modular program specifications with history information by providing separation over both space (program states) and time. We verify Michael's non-blocking stack algorithm and show that the intuition behind such algorithm can be naturally captured using trace invariants.

1 Introduction

Optimistic concurrency algorithms [6, 7] allow concurrent access to shared data and ensure data consistency by performing dynamic conflict detection. These algorithms can be more efficient than coarse-grained lock-based synchronization if there is sufficient data independence. However, the design of the algorithms has to consider many more thread-interleaving scenarios than coarse-grained synchronization. The algorithms are usually complex and error-prone. Their correctness is usually far from obvious and is hard to verify too.

As an example, Fig. 1 shows a non-blocking stack algorithm, where a stack is implemented as a linked list pointed by the `Top` pointer. It allows simultaneous read (line 4 and 13) and write (7, 15) of `Top`, and the conflict detection is done by the CAS (compare-and-swap) command. This algorithm has two subtle bugs. One is that `t` might be a dangling pointer when the dereference occurs in line `6`. The other is the notorious ABA problem: suppose the top three nodes on the stack are A, B and C; Thread 1 calls `pop` and reaches the end of line `6`; so `t` points to A and `next` points to B; then Thread 2 comes, pops A and B, and pushes A onto the stack; Thread 1 continues to execute line `7`, where the comparison succeeds and `Top` is set to point to B, which is no longer on the stack.

Here, we have to refer to the historical events to explain the problems above. It is not surprising that temporal reasoning is needed to argue for the correctness of such highly concurrent algorithms.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 388–402, 2010.

```
pop(){
 local done, next, t;
 done := false;
 while (!done){
   t := Top;
   if (t == null) return null;
   next := t.Next;
   done := CAS(&Top, t, next);
 }
 return t;
}
```

```
push(x){
local done, t;
 done := false;
 while (!done){
   t := Top;
   x.Next := t;
   done := CAS(&Top, t, x);
 }
 return true;
}
```

Fig. 1. A Buggy Implementation of Non-Blocking Stacks

Concurrent separation logic (CSL [13]) and rely-guarantee (R-G) based reasoning [8] are two well-studied approaches to concurrency verification. Previous work [14, 18] has shown that they can be used to verify fine-grained and optimistic algorithms. However, since assertions in these logics only specify program states (or state transitions in R-G reasoning), it is difficult to use them to express directly the temporal properties about the subtle interaction between threads. Instead, we have to introduce history variables to record the occurrence of certain events. This indirect approach to specifying historical events makes specifications and proofs complex, and in many cases fails to demonstrate the high-level intuition behind the design of the algorithms.

In this paper, we propose a new program logic that uses invariants on historical execution traces to reason about optimistic concurrency algorithms. The logic extends previous work on R-G reasoning by introducing past tense temporal operators in the assertion language. It allows us to specify historical events directly without using history variables, which makes the verification process more modular and intuitive.

Although it has also been observed before that past tense operators in temporal logic can be used to eliminate the need of history variables [10], developing a modular logic with temporal reasoning that is able to verify modern concurrent algorithms has so far been an open problem. Our logic inherits previous work on combining R-G reasoning with separation logic [3, 19, 2] to support modular verification. Separating conjunction in separation logic is now defined over assertions on execution histories instead of state assertions. The frame rule and the hide rule in the Local Rely-Guarantee (LRG) logic [2]—the keys for modular verification—are supported naturally in this new setting.

We apply our new logic to reason about Michael's non-blocking stack algorithm [11] which uses hazard pointers to fix the buggy algorithm in Fig. 1. We use trace invariants to capture the main intuition underlying the algorithm. The program specifications and proofs used in our logic are more intuitive than those from previous work [14]. They do not require history variables. Our logic also supports a new frame rule that further simplifies the proofs (e.g., for the `retireNode` function in Fig. 9) and makes the verification more modular.

$$\begin{array}{rll}
(\mathsf{Expr}) & E ::= x \mid n \mid E\texttt{+}E \mid E\texttt{-}E \mid \ \ldots & \\
(\mathsf{Bexp}) & B ::= \mathsf{true} \mid \mathsf{false} \mid E\texttt{=}E \mid E\neq E \mid \ \ldots & \\
(\mathsf{Stmts}) & C ::= x := E \mid x := \texttt{[}E\texttt{]} \mid \texttt{[}E\texttt{]} := E' \mid \mathbf{skip} \mid x := \mathbf{cons}(E, \ldots, E) & \\
 & \quad \mid \mathbf{dispose}(E) \mid \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C \mid \mathbf{while}\ B\ \mathbf{do}\ C \mid \langle C \rangle \mid C;C & \\
(\mathsf{Prog}) & W ::= \ \mathsf{t}_1.C_1 \,\|\, \ldots \,\|\, \mathsf{t}_n.C_n & (\mathsf{ThrdID}) \quad \mathsf{t} \ \in \ \mathsf{Nat}
\end{array}$$

Fig. 2. A Concurrent Programming Language

$$\begin{array}{llll}
(\mathsf{Store}) \ s \ \in \ \mathsf{PVar} \rightharpoonup_{\mathrm{fin}} \mathsf{Int} & & (\mathsf{Heap}) \ h \ \in \ \mathsf{Nat} \rightharpoonup_{\mathrm{fin}} \mathsf{Int} \\
(\mathsf{State}) \ \sigma \ \in \ \mathsf{Store} \times \mathsf{Heap} & & \\
(\mathsf{Trace}) \ \mathcal{T} \ ::= \ (\sigma_0, \mathsf{t}_0) :: (\sigma_1, \mathsf{t}_1) :: \cdots :: (\sigma_n, \mathsf{t}_n) & & (\mathsf{Trans}) \ \mathcal{R}, \mathcal{G} \ \in \ \mathcal{P}(\mathsf{Trace})
\end{array}$$

Fig. 3. Program States and Execution Traces

$$\frac{\mathcal{T}.\mathit{last} = (\sigma, \_) \quad (C, \sigma) \longrightarrow (C', \sigma')}{(\mathsf{t}.C, \mathcal{T}) \hookrightarrow (\mathsf{t}.C', \mathcal{T} :: (\sigma', \mathsf{t}))} \qquad \frac{\mathcal{T}.\mathit{last} = (\sigma, \_) \quad (C, \sigma) \longrightarrow \mathbf{abort}}{(\mathsf{t}_i.C_i, \mathcal{T}) \hookrightarrow \mathbf{abort}}$$

$$\frac{(\mathsf{t}_i.C_i, \mathcal{T}) \hookrightarrow (\mathsf{t}_i.C_i{}', \mathcal{T}')}{(\mathsf{t}_1.C_1 \,\|\, \ldots \mathsf{t}_i.C_i \ldots \,\|\, \mathsf{t}_n.C_n, \mathcal{T}) \overset{\mathcal{R}}{\longmapsto} (\mathsf{t}_1.C_1 \,\|\, \ldots \mathsf{t}_i.C_i{}' \ldots \,\|\, \mathsf{t}_n.C_n, \mathcal{T}')}$$

$$\frac{(\mathsf{t}_i.C_i, \mathcal{T}) \hookrightarrow \mathbf{abort}}{(\mathsf{t}_1.C_1 \,\|\, \ldots \mathsf{t}_i.C_i \ldots \,\|\, \mathsf{t}_n.C_n, \mathcal{T}) \overset{\mathcal{R}}{\longmapsto} \mathbf{abort}} \qquad \frac{(\mathcal{T} :: (\sigma, \mathsf{t})) \in \mathcal{R}}{(W, \mathcal{T}) \overset{\mathcal{R}}{\longmapsto} (W, \mathcal{T} :: (\sigma, \mathsf{t}))}$$

Fig. 4. Selected Rules of Operational Semantics

2 A Concurrent Programming Language

Figure 2 shows a simple concurrent language. The statements $x := \texttt{[}E\texttt{]}$ and $\texttt{[}E\texttt{]} := E$ are memory-load and store operations respectively; **cons** allocates fresh memory cells, and **dispose** frees a cell. The atomic block $\langle C \rangle$ executes C atomically. Other statements have standard meanings. A program W contains n parallel threads, each marked with a unique thread ID (e.g., t_i for C_i).

Figure 3 defines program states and execution traces. The store s is a finite partial mapping from program variables to integers; the heap h maps memory locations (natural numbers) to integers. A program state σ is a pair (s, h). An execution trace $\mathcal{T}$ is a (nonempty) finite sequence $(\sigma_0, \mathsf{t}_0) :: (\sigma_1, \mathsf{t}_1) :: \cdots :: (\sigma_n, \mathsf{t}_n)$. A pair (σ_i, t_i) in a trace $\mathcal{T}$ means that a thread with ID t_i reached the state σ_i after executing one step from the state σ_{i-1}. Thread ID t_0 can be any value. We use $\mathcal{T}.\mathit{last}$ to denote the last element in $\mathcal{T}$. A trace $(\mathcal{T} :: (\sigma, \mathsf{t}))$ in the trace sets $\mathcal{R}$ and $\mathcal{G}$ is also used to model a single-step transition by the thread t that starts from $\mathcal{T}$ and reaches a new state σ.

$$\begin{array}{ll}
(\mathsf{StateAssert}) & P, Q ::= B \mid \mathsf{emp}_h \mid \mathsf{emp}_s \mid \mathsf{own}(x) \mid E \mapsto E \mid P * Q \mid \ldots \\
(\mathsf{TraceAssert}) & p, q, R, G, I ::= P \mid \mathsf{Id} \mid [p]_{\mathsf{t}} \mid p \rhd q \mid \ominus p \mid p * q \mid \exists X.p \mid \neg p \mid p \vee q \mid \ldots
\end{array}$$

Fig. 5. The Assertion Language

$$\begin{array}{lll}
(s,h) \models_{\mathrm{SL}} \mathsf{emp}_s & \text{iff} & s = \emptyset \qquad\qquad (s,h) \models_{\mathrm{SL}} \mathsf{emp}_h \quad \text{iff} \quad h = \emptyset \\
(s,h) \models_{\mathrm{SL}} \mathsf{own}(x) & \text{iff} & \mathsf{dom}(s) = \{x\} \\
(s,h) \models_{\mathrm{SL}} E_1 \mapsto E_2 & \text{iff} & \text{there exist } \ell \text{ and } n \text{ such that} \\
 & & \qquad [\![E_1]\!]_s = \ell, [\![E_2]\!]_s = n, \mathsf{dom}(h) = \{\ell\} \text{ and } h(\ell) = n \\
\sigma \models_{\mathrm{SL}} P * Q & \text{iff} & \text{there exist } \sigma_1 \text{ and } \sigma_2 \\
 & & \qquad \text{such that } \sigma_1 \uplus \sigma_2 = \sigma, \sigma_1 \models_{\mathrm{SL}} P \text{ and } \sigma_2 \models_{\mathrm{SL}} Q \\
\mathsf{emp} & \stackrel{\text{def}}{=} & \mathsf{emp}_s \wedge \mathsf{emp}_h
\end{array}$$

Fig. 6. Semantics of Selected Separation Logic Assertions

Figure 4 gives selected rules modeling the operational semantics (see the technical report [4] for the complete rules). The binary relation $\_ \longrightarrow \_$ models a transition over states made by a primitive statement. The definition is standard and is omitted here. The relation $\_ \hookrightarrow \_$ lifts state transitions to transitions over traces. The thread ID of the executing thread t is recorded on the trace at the end of the transition. Finally, the relation $\_ \xmapsto{\mathcal{R}} \_$ models the transitions over traces made by programs in an environment characterized by $\mathcal{R}$. Here $\mathcal{R}$ contains all the possible transitions of the environment.

3 The Assertion Language

Our assertion language is defined in Fig. 5. We use separation logic assertions (P and Q) to specify program states. Following Parkinson et al. [15], we also treat program variables as resources. Semantics of some separation logic assertions are shown in Fig. 6. We use $\sigma_1 \uplus \sigma_2$ to represent the union of the two disjoint states.

Trace assertions. Trace assertions p, q, R, G and I specify historical execution traces. Semantics of trace assertions are defined in Fig. 7. Here we use $|\mathcal{T}|$ to represent the length of $\mathcal{T}$, and use $\mathcal{T}_{k-}$ to represent the subsequence resulting from truncating the last k elements from $\mathcal{T}$ $(0 \leq k < |\mathcal{T}|)$.

A state assertion P is viewed as a trace assertion that specifies only the last state. Assertion Id says that the last two states on the trace are the same (i.e. the last state transition is an identity transition). Assertion $[p]_{\mathsf{t}}$ means that the trace satisfies p and the last state transition is made by the thread t. Assertion $p \rhd q$ holds over $\mathcal{T}$ if and only if p holds over the trace $\mathcal{T}_{i-}$ for some i and q holds ever since. It is also represented as $q\ \mathcal{S}\ p$ (q since p) in the literature of temporal logic [10]. Assertion $\ominus p$ holds if and only if the trace prior to the last transition satisfies p. $[\![p]\!]$ is the set of traces that satisfy p.

$$
\begin{array}{lll}
\mathcal{T}_{k-} & \stackrel{\text{def}}{=} & (\sigma_0, \mathsf{t}_0) :: \cdots :: (\sigma_{n-k}, \mathsf{t}_{n-k}) \quad \text{if } \mathcal{T} = (\sigma_0, \mathsf{t}_0) :: \cdots :: (\sigma_n, \mathsf{t}_n) \text{ and } 0 \le k \le n \\
\mathcal{T} \models P & \text{iff} & \text{there exists } \sigma \text{ such that } \mathcal{T}.last = (\sigma, \_) \text{ and } \sigma \models_{\text{SL}} P \\
\mathcal{T} \models \mathsf{Id} & \text{iff} & \text{there exist } \mathcal{T}' \text{ and } \sigma \text{ such that } \mathcal{T} = \mathcal{T}' :: (\sigma, \_) :: (\sigma, \_) \\
\mathcal{T} \models [p]_{\mathsf{t}} & \text{iff} & \mathcal{T}.last = (\_, \mathsf{t}) \text{ and } \mathcal{T} \models p \\
\mathcal{T} \models p \triangleright q & \text{iff} & \text{there exists } 0 < i < |\mathcal{T}| \text{ such that } \mathcal{T}_{i-} \models p \text{ and } \forall j < i.\, \mathcal{T}_{j-} \models q \\
\mathcal{T} \models \ominus p & \text{iff} & 1 < |\mathcal{T}| \text{ and } \mathcal{T}_{1-} \models p \qquad\qquad [\![p]\!] \stackrel{\text{def}}{=} \{\mathcal{T} \mid \mathcal{T} \models p\}
\end{array}
$$

$$
((\sigma_0, \mathsf{t}_0) :: \ldots :: (\sigma_n, \mathsf{t}_n)) \oplus ((\sigma'_0, \mathsf{t}'_0) :: \ldots :: (\sigma'_m, \mathsf{t}'_m))
$$

$$
\stackrel{\text{def}}{=} \begin{cases} ((\sigma_0 \uplus \sigma'_0, \mathsf{t}_0) :: \ldots :: (\sigma_n \uplus \sigma'_m, \mathsf{t}_n)) & \text{if } n = m \wedge \forall 0 \le i \le n. \mathsf{t}_i = \mathsf{t}'_i \\ \text{undefined} & \text{otherwise} \end{cases}
$$

$$
\mathcal{T} \models p * q \quad \text{iff} \quad \text{there exist } \mathcal{T}_1 \text{ and } \mathcal{T}_2 \text{ such that } \mathcal{T} = \mathcal{T}_1 \oplus \mathcal{T}_2, \mathcal{T}_1 \models p \text{ and } \mathcal{T}_2 \models q
$$

$$
\begin{array}{llllll}
p \trianglerighteq q & \stackrel{\text{def}}{=} (p \triangleright q) \vee p & \diamondminus\, p & \stackrel{\text{def}}{=} p \trianglerighteq \mathsf{true} & \boxminus\, p & \stackrel{\text{def}}{=} \neg \diamondminus\, (\neg p) \\
p \blacktriangleright q & \stackrel{\text{def}}{=} \diamondminus\, (\diamondminus\, p \wedge q) & p \ltimes_{\mathsf{t}} q & \stackrel{\text{def}}{=} \ominus p \wedge [q]_{\mathsf{t}} & p \ltimes q & \stackrel{\text{def}}{=} \exists \mathsf{t}.\, p \ltimes_{\mathsf{t}} q
\end{array}
$$

We assume unary operators ($\diamondminus$ and $\ominus$) have higher precedence than other operators.

Fig. 7. Semantics of Trace Assertions

Assertion $p * q$ lifts separating conjunction to traces; it specifies a program trace consisting of two *disjoint* parts: one satisfies p and another q. Traces $\mathcal{T}_1$ and $\mathcal{T}_2$ are disjoint if they have the same length, and for each i such that $0 \le i < |\mathcal{T}|$ the states in $\mathcal{T}_1[i]$ and $\mathcal{T}_2[i]$ are disjoint (see the definition of $\mathcal{T}_1 \oplus \mathcal{T}_2$ in Fig. 7).

Other useful connectors can be defined using these primitive operators. Assertion $p \trianglerighteq q$ is a weaker version of $p \triangleright q$. Assertion $\diamondminus\, p$ says that p was once true in the history. Assertion $\boxminus p$ holds if and only if p holds at every step in the history. Assertion $p \blacktriangleright q$ says that p first came true in the history, and then q came true later. Assertion $p \ltimes_{\mathtt{t}} q$ means that the last transition is made by thread $\mathtt{t}$, and assertion p holds prior to the transition, and q holds after it. This allows us to define the built-in $\ltimes$ operator in LRG [2].

Example 3.1. In the TL2 transactional memory protocol [1], before updating a shared memory cell, we must first acquire the corresponding lock and then increase the global version clock. This requirement (among many others in the protocol) can be defined as the following guarantee:

$$
G_{\mathtt{tid}}(\mathtt{x}) \stackrel{\text{def}}{=} \exists D, D', T, T'. \begin{pmatrix} ((\mathtt{x} \mapsto 0, D * \mathtt{gt} \mapsto T) \triangleright (\mathtt{x} \mapsto \mathtt{tid}, D * \mathtt{gt} \mapsto \_)) \\ \wedge (\mathtt{x} \mapsto \mathtt{tid}, D * \mathtt{gt} \mapsto T') \wedge (T' > T) \end{pmatrix} \\ \ltimes_{\mathtt{tid}} (\mathtt{x} \mapsto \mathtt{tid}, D' * \mathtt{gt} \mapsto T')
$$

Here $\mathtt{x}$ points to two fields, its lock and its value. The first line above says that, before the transition, the lock was acquired (it was changed from 0 to $\mathtt{tid}$) when the global version clock $\mathtt{gt}$ was T. Then the lock and the value have been preserved ever since, but $\mathtt{gt}$ might have been changed. The second line says $\mathtt{gt}$

is greater than T right before the transition. The third line says the value of x is updated by the transition. This definition also implies that the increment of gt is done after the lock is acquired.

The guarantee above refers to two events before the specified transition. In traditional R-G reasoning, the guarantee condition can only specify two states, so we have to introduce history variables to describe such historical events.

As in separation logic, a class of trace assertions that are of special interest to us are those that are precise about the last state on the trace.

Definition 3.2 (Last-State-Precise Trace Assertions). *p is last state precise,* i.e. $\mathsf{LPrec}(p)$ *holds, if and only if for all $\mathcal{T}$,* t*, s, h, s_1, s_2, h_1, h_2, if $s_1 \subseteq s$, $s_2 \subseteq s$, $h_1 \subseteq h$, $h_2 \subseteq h$, $\mathcal{T} :: ((s_1, h_1), \mathsf{t}) \models p$ and $\mathcal{T} :: ((s_2, h_2), \mathsf{t}) \models p$, then $s_1 = s_2$ and $h_1 = h_2$.*

The example below shows a last-state-precise assertion p can specify a precise state domain that is determined dynamically by historical events. It is more powerful than a precise state assertion P in separation logic [17]. This can also be seen in our hazard-pointer-based stack example.

Example 3.3. Let $I = \exists X. \diamondminus (\ell \mapsto X * \mathsf{true}) \wedge (\ell \mapsto \_ * (X = 0 \wedge r \vee X \neq 0 \wedge \mathsf{emp}))$ where $r = x \mapsto \_ * y \mapsto \_$, then I is a last-state-precise trace assertion. It specifies traces where the domain of the last state depends on the historical value X of ℓ.

4 A Program Logic for History

Now we present our program logic for history, named HLRG, which extends the LRG logic [2] with trace assertions for reasoning about historical traces.

As in LRG, we use the judgments $R; G \vdash \{p\} W \{q\}$ and $R; G; I \vdash_{\mathsf{t}} \{p\} C \{q\}$ for well-formed programs and well-formed thread t respectively. The rely condition R and the guarantee G specify the interference between the environment and the thread. The judgments say informally that starting from a trace satisfying both $\boxminus(R \vee G) * \mathsf{true}$ and p, if the environment's transitions satisfy R, then W (or C) would not abort, its transitions satisfy G, and q holds at the end if W (or C) terminates. The invariant I specifies the well-formedness of the shared resource. Unlike in the LRG logic, R, G, I, p and q are all trace assertions now.

Figure 8 gives the main inference rules. The PROG rule allows us to verify the whole program by verifying the n parallel threads $\mathsf{t}_1.C_1$,$\mathsf{t}_2.C_2$,. . ., $\mathsf{t}_n.C_n$ separately. Each thread t_i has exclusive access to its own private resource specified by p_i and q_i. All threads can access the shared resource specified by r, $r_1 \ldots r_n$. To verify each thread, we need to find an invariant I specifying the basic well-formedness of the shared resource.

The ATOM rule says that we can treat C in the atomic block as sequential code since its execution cannot be interrupted. Here the judgment $\{P\}C\{Q\}$ can be derived following the standard sequential separation logic rules [17], which we do not show here. This rule allows us to strengthen P into a trace assertion p so

$$\frac{\begin{array}{c} R\vee G_1\vee\ldots\vee G_{i-1}\vee G_{i+1}\vee\ldots\vee G_n; G_i; I\vdash_{\mathsf{t}_i}\{p_i * r\}C_i\{q_i * r_i\} \quad \forall i\in\{1,\ldots,n\} \\ r\vee r_1\vee\ldots\vee r_n \Rightarrow I \end{array}}{R; G_1\vee\ldots\vee G_n\vdash\{p_1*\ldots*p_n*r\}\mathsf{t}_1.C_1 \,\|\, \ldots \,\|\, \mathsf{t}_n.C_n\{q_1*\ldots*q_n*(r_1\wedge\ldots\wedge r_n)\}} \ (\textsc{prog})$$

$$\frac{p\Rightarrow P \quad \{P\}C\{Q\} \quad p\ltimes_{\mathsf{t}} Q\Rightarrow G*\mathsf{true}}{\mathsf{Id}_I; G; I\vdash_{\mathsf{t}}\{p\}\,\langle C\rangle\,\{Q\}} \ (\textsc{atom})$$

where Id_I is defined as $\mathsf{Id}\wedge(I\ltimes I)$.

$$\frac{p\Rightarrow p' \quad \mathsf{Id}_I; G; I\vdash_{\mathsf{t}}\{p'\}\langle C\rangle\{Q'\} \quad \ominus p\wedge Q'\Rightarrow q \quad \mathtt{Sta}(\{p,q\}, R*\mathsf{Id})}{R; G; I\vdash_{\mathsf{t}}\{p\}\,\langle C\rangle\,\{q\}} \ (\textsc{atom-r})$$

$$\frac{\begin{array}{c} R; G; I\vdash_{\mathsf{t}}\{p\}C\{q\} \\ \mathtt{Sta}(r, R'*\mathsf{Id}) \quad r\Rightarrow I'*\mathsf{true} \end{array}}{R*R'; G*G'; I*I'\vdash_{\mathsf{t}}\{p*r\}C\{q*r\}} \ (\textsc{frame}) \quad \frac{R; G; I\vdash_{\mathsf{t}}\{p\}C\{q\}}{R; G; I\vdash_{\mathsf{t}}\{p\wedge\diamondminus r\}C\{q\wedge\diamondminus r\}} \ (\textsc{frameT})$$

$$\frac{R; G; I\vdash_{\mathsf{t}}\{p\wedge(I'*\mathsf{true})\}C\{q\} \quad \boxminus(R\vee G)\Rightarrow(I'\wedge I'')}{R; G; I\vdash_{\mathsf{t}}\{p\}C\{q\wedge(I''*\mathsf{true})\}} \ (\textsc{inv})$$

Fig. 8. Selected Inference Rules of the HLRG Program Logic

that we can carry the historical information. The transition from p to Q needs to satisfy the guarantee G, which may have some constraints over the history traces (examples about G can be found in Fig. 10 in Sec. 5).

The ATOM rule uses a strong rely condition about the environment, which is an identity transition preserving the invariant I of the shared resource. To relax it, we can apply the next ATOM-R rule borrowed from RGSep [18]. It allows us to adjust the pre- and post-conditions so that they are both *stable* with respect to the rely condition R.

Definition 4.1 (Stability). *We say a trace assertion p is stable with respect to a trace assertion q,* i.e. $\mathtt{Sta}(p,q)$ *holds, if and only if* $\ominus p\wedge q\Rightarrow p$.

That is, if p holds before the most recent transition, and the transition satisfies q, then p holds after it. This is the standard requirement in R-G reasoning. With temporal operators, it can now be encoded as a temporal assertion. We use $\mathtt{Sta}(\{p,q\},R)$ as a shorthand for $\mathtt{Sta}(p,R)\wedge\mathtt{Sta}(q,R)$.

The interesting (and *new*) part of this ATOM-R rule is the post condition q, which is weakened from the trace assertion $\ominus p\wedge Q'$. This allows us to carry information about historical events happened before this atomic transition.

The FRAME rule comes from LRG. It supports local reasoning and allows us to write "small" specifications about resources that are indeed accessed in C. Invariants about other resources are preserved and can be added into the specifications later. We also introduce a new FRAMET rule to show the frame property over "time". Since historical traces would not affect the execution of programs, knowledge about history can be added when necessary.

The new INV rule is also very useful. It is like the reverse of the standard consequence rule in Hoare logic, since it allows us to strengthen the pre-condition, and prove a post-condition weaker than we wanted. This rule is sound because the invariants I' and I'' can be derived from the fact that each step of the transition satisfies $R \vee G$, so that they can be used anywhere in the proof for free. We will demonstrate the use of FRAME, FRAMET and INV in our example in Sec. 5.

The rest of the rules are the same as those in LRG, and are not shown here. Note that in each rule we implicitly require the following properties hold.

- $\mathsf{fence}(I, R)$ and $\mathsf{fence}(I, G)$;
- $p \vee q \Rightarrow I * \mathsf{true}$;

where $\mathsf{fence}(I, p)$ is defined below:

$$\mathsf{fence}(I, p) \stackrel{\text{def}}{=} (\mathsf{Id} \wedge (I \ltimes I) \Rightarrow p) \wedge (p \Rightarrow I \ltimes I) \wedge \mathsf{LPrec}(I)\,.$$

Informally, it requires that the most recent transition is confined in a precise domain enforced by the last-state-precise assertion I. This constraint is inherited from LRG. Interested readers can refer to our previous work [2] to see the technical discussions about this requirement.

Semantics and soundness. The semantics of our logic and its soundness proof are similar to those of LRG. We first define the non-interference property below.

Definition 4.2 (Non-Interference). *Let* $W = \mathsf{t}_1.C_1 \,\|\, \ldots \,\|\, \mathsf{t}_n.C_n$. $(W, \mathcal{T}, \mathcal{R}) \Longrightarrow^0 \mathcal{G}$ *always holds.* $(W, \mathcal{T}, \mathcal{R}) \Longrightarrow^{m+1} \mathcal{G}$ *holds iff* $\neg(W, \mathcal{T}) \xmapsto{\mathcal{R}}$ **abort** *and the following are true:*

1. *for all* t *and* σ, *if* $(\mathcal{T} :: (\sigma, \mathsf{t})) \in \mathcal{R}$, *then for all* $k \leq m$, $(W, \mathcal{T} :: (\sigma, \mathsf{t}), \mathcal{R}) \Longrightarrow^k \mathcal{G}$;
2. *for all* σ *and* $i \in \{1, \ldots, n\}$, *if* $(\mathsf{t}_i.C_i, \mathcal{T}) \hookrightarrow (\mathsf{t}_i.C_i{}', \mathcal{T} :: (\sigma, \mathsf{t}_i))$, *then* $(\mathcal{T} :: (\sigma, \mathsf{t}_i)) \in \mathcal{G}$ *and* $(\mathsf{t}_1.C_1 \,\|\, \ldots \mathsf{t}_i.C_i{}' \ldots \,\|\, \mathsf{t}_n.C_n, \mathcal{T} :: (\sigma, \mathsf{t}_i), \mathcal{R}) \Longrightarrow^k \mathcal{G}$ *holds for all* $k \leq m$.

Then the semantics of $R; G \vdash \{p\}W\{q\}$ is defined below. Theorem 4.4 shows the soundness theorem of the logic.

Definition 4.3. $R; G \models \{p\}W\{q\}$ *iff, for all* $\mathcal{T}$ *such that* $\mathcal{T} \models p \wedge (\boxminus(R \vee G) * \mathsf{true})$, *the following are true (where* $\mathcal{R} = [\![R{*}\mathsf{Id}]\!]$ *and* $\mathcal{G} = [\![G{*}\mathsf{true}]\!]$*):*

1. *if* $(W, \mathcal{T}) \xmapsto{\mathcal{R}}{}^{*} (\mathbf{skip}, \mathcal{T}')$, *then* $\mathcal{T}' \models q$;
2. *for all* m, $(W, \mathcal{T}, \mathcal{R}) \Longrightarrow^m \mathcal{G}$.

Theorem 4.4 (Soundness). *If* $R; G \vdash \{p\}W\{q\}$ *then* $R; G \models \{p\}W\{q\}$.

We show the proof in the extended technical report [4].

```
pop(){
01  local done, next, t, t1;
02  done := false;
03  while (!done){
04    t := Top;
05    if (t == null) return null;
06    HP[tid] := t;
07    t1 := Top;
08    if (t == t1){
09      next := t.Next;
10      done := CAS(&Top, t, next);
11    }
12  }
13  retireNode(t);
14  HP[tid] := null;
15  return t;
}
```

```
retireNode(t){
16  local i, t';
17  i := 1;
18  while(i<=th_num){
19    if (i != tid){
20      t' := HP[i];
21      if (t'!= t){
22        i:= i+1;
23      }
24    }else i:= i+1;
25  }
}
```

Fig. 9. Optimistic Lock-Free Stacks with Hazard Pointers

5 Verification of Lock-Free Stacks with Hazard Pointers

We now apply HLRG to verify Michael's lock-free stacks, which use hazard pointers [11] to address the problems with the algorithm in Fig. 1. In Fig. 9 we show the new pop function. The push function is the same as in Fig. 1 and is omitted here. We use $\mathsf{stack}(\mathtt{Top}, A)$ below to specify the shared stack, which is implemented as a linked list pointed by Top. The set A records the memory locations of the nodes on the list. It is kept to simplify our proofs. Below we use $E \mapsto E_1, E_2$ as a shorthand for $E \mapsto E_1 * E{+}1 \mapsto E_2$, and $E \mapsto \_$ for $\exists n.\, E \mapsto n$.

$$\begin{aligned}
\mathsf{List}(\ell, \emptyset, \mathsf{nil}) &\stackrel{\text{def}}{=} \mathsf{emp} \wedge \ell = \mathbf{null} \\
\mathsf{List}(\ell, A, n{::}L) &\stackrel{\text{def}}{=} \ell \in A \wedge \exists \ell'.\, (\ell \mapsto n, \ell') * \mathsf{List}(\ell', A{-}\{\ell\}, L) \\
\mathsf{stack}(\mathtt{Top}, A) &\stackrel{\text{def}}{=} \exists \ell, L.\, (\mathtt{Top} \mapsto \ell) * \mathsf{List}(\ell, A, L)
\end{aligned} \tag{1}$$

The algorithm fixes the ABA problem by using a global array HP, which contains a "hazard" pointer for each thread. The array is specified by $I_{\mathrm{hp}}(\mathtt{HP})$. Here HP+tid is the location of HP[tid], and th_num is the number of threads.

$$I_{\mathrm{hp}}(\mathtt{HP}) \stackrel{\text{def}}{=} \circledast_{\mathtt{tid} \in [1..\mathtt{th\_num}]}.\, \mathtt{HP}{+}\mathtt{tid} \mapsto \_ \tag{2}$$

where $\circledast_{x \in s}.p(x) \stackrel{\text{def}}{=} s = \emptyset \wedge \mathsf{emp} \vee \exists z.\, (s = \{z\} \uplus s') \wedge (\circledast_{x \in s'}.p(x)) * p(z)$
and $\uplus$ is the union of disjoint sets.

Before a racy access to the top node on the stack, a thread stores the node's memory location into its HP entry (lines 06-08). This announces to other threads that the node is being accessed and should not be reclaimed. When a node is successfully removed from the stack (line 10), the remover thread calls retireNode (line 13) and waits till after this node is no longer being accessed by any other threads (i.e., not pointed by their HP entries). Finally, it clears its own HP entry (line 14) before it obtains the full ownership of the node (line 15).

We use $\mathsf{remove}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid})$ in (3) to specify that the thread tid is in the *remove* phase: it has popped the node at ℓ from the stack, but has not reached line 14 yet. The part in front of $\trianglerighteq$ says that there was a primitive operation in history, which popped the node from the stack. The $\trianglerighteq$ operator and the assertion following it require that the removed node be pointed by the remover's own HP entry ever since. Here $E_1 \rightsquigarrow E_2$ is a shorthand for $(E_1 \mapsto E_2) * \mathsf{true}$. The predicate $\mathsf{not\_rem}(\mathtt{Top}, \mathtt{HP}, \mathtt{tid})$ in (4) says that tid is currently not in the remove phase.

$$\begin{aligned}&\mathsf{remove}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}) \stackrel{\mathrm{def}}{=}\\ &\quad \big(((\mathtt{HP}{+}\mathtt{tid} \mapsto \ell * \mathtt{Top} \rightsquigarrow \ell) \ltimes_{\mathtt{tid}} (\mathtt{HP}{+}\mathtt{tid} \mapsto \ell * \exists \ell'.\mathtt{Top} \rightsquigarrow \ell' \wedge \ell \neq \ell'))\\ &\qquad\qquad \trianglerighteq \mathtt{HP}{+}\mathtt{tid} \rightsquigarrow \ell\big) \wedge \ell \neq \mathtt{null}\end{aligned} \tag{3}$$

$$\mathsf{not\_rem}(\mathtt{Top}, \mathtt{HP}, \mathtt{tid}) \stackrel{\mathrm{def}}{=} \neg \exists \ell.\, \mathsf{remove}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}) \tag{4}$$

In addition to the stack and the HP array, the popped nodes that are accessible from the hazard pointers should be viewed as shared resources as well. We use $\mathsf{opset}(\mathtt{Top}, \mathtt{HP}, S, O)$ in (5) to iterate these shared nodes, where O is the set of pointers pointing to these nodes and S is the set of threads.

$$\begin{aligned}&\mathsf{opset}(\mathtt{Top}, \mathtt{HP}, \emptyset, \emptyset) \stackrel{\mathrm{def}}{=} \mathsf{true}\\ &\mathsf{opset}(\mathtt{Top}, \mathtt{HP}, \{\mathtt{tid}\} \uplus S, O) \stackrel{\mathrm{def}}{=}\\ &\qquad (\exists \ell.\, \mathsf{remove}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}) \wedge \ell \in O \wedge \mathsf{opset}(\mathtt{Top}, \mathtt{HP}, S, O - \{\ell\}))\\ &\quad \vee (\mathsf{not\_rem}(\mathtt{Top}, \mathtt{HP}, \mathtt{tid}) \wedge \mathsf{opset}(\mathtt{Top}, \mathtt{HP}, S, O))\end{aligned} \tag{5}$$

The invariant I below specifies all three parts of the shared resources. I is a last-state-precise assertion. The domain of the shared resource depends on the historical information whether nodes are popped before or not.

$$\begin{aligned}I \stackrel{\mathrm{def}}{=} \exists O.\, &\mathsf{opset}(\mathtt{Top}, \mathtt{HP}, [1..\mathtt{th\_num}], O)\\ &\wedge (I_{\mathrm{hp}}(\mathtt{HP}) * \exists A.\mathsf{stack}(\mathtt{Top}, A) * (\circledast_{\ell \in O}.\, \ell \mapsto \_, \_))\end{aligned} \tag{6}$$

Below we characterize the meaning of hazard pointers. $\mathsf{ishazard}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid})$ says $\mathtt{HP}[\mathtt{tid}]$ contains a "confirmed" hazard pointer ℓ, i.e. ℓ was once the top of the stack in history and the thread tid has not updated the Top pointer ever since (though Top might have been updated by other threads). When the remover thread invokes retireNode on the top node t, it scans the hazard pointers of all other threads and make sure that $\mathsf{ishazard}(\mathtt{t}, \mathtt{Top}, \mathtt{HP}, \mathtt{tid})$ does not hold for each non-remover thread tid. This is specified by $\mathsf{hazfree}(\mathtt{t}, \mathtt{Top}, \mathtt{HP}, \mathtt{tid})$, which says that the node t has been popped by the thread tid and other threads no longer treat it as a hazard node.

$$\mathsf{upd\_top}(\mathtt{tid}) \stackrel{\mathrm{def}}{=} \exists \ell, \ell'.\, (\mathtt{Top} \rightsquigarrow \ell \ltimes_{\mathtt{tid}} \mathtt{Top} \rightsquigarrow \ell') \wedge \ell \neq \ell'$$

$$\begin{aligned}&\mathsf{ishazard}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}) \stackrel{\mathrm{def}}{=}\\ &\quad (\mathtt{HP}{+}\mathtt{tid} \mapsto \ell * \mathtt{Top} \rightsquigarrow \ell) \triangleright \big((\mathtt{HP}{+}\mathtt{tid} \rightsquigarrow \ell) \wedge \neg \mathsf{upd\_top}(\mathtt{tid})\big)\end{aligned} \tag{7}$$

$$\begin{aligned}&\mathsf{hazfree}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}) \stackrel{\mathrm{def}}{=}\\ &\quad \mathsf{remove}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}) \wedge \forall \mathtt{tid}' \in [1..\mathtt{th\_num}].\, \mathtt{tid}' = \mathtt{tid} \vee \neg \mathsf{ishazard}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}')\end{aligned} \tag{8}$$

$$\begin{array}{lll}
\mathsf{Pop}_{\texttt{tid}} & \overset{\text{def}}{=} \exists \ell, \ell'. \ominus ((\texttt{Top} \rightsquigarrow \ell) * (\texttt{HP}+\texttt{tid} \mapsto \ell) * (\ell \mapsto \_, \ell') * \mathsf{List}(\ell', \_, \_)) & \\
& \quad \wedge((\texttt{Top} \mapsto \ell \ltimes_{\texttt{tid}} \texttt{Top} \mapsto \ell') * \mathsf{Id}) & (\text{line } 10) \\
\mathsf{Retire}_{\texttt{tid}} & \overset{\text{def}}{=} \exists \ell. \ominus \mathsf{hazfree}(\ell, \texttt{Top}, \texttt{HP}, \texttt{tid}) & \\
& \quad \wedge (((\texttt{HP}+\texttt{tid} \mapsto \ell * \ell \mapsto \_, \_) \ltimes_{\texttt{tid}} \texttt{HP}+\texttt{tid} \mapsto \texttt{null}) * \mathsf{Id}) & (\text{line } 14) \\
\mathsf{Reset\_HP}_{\texttt{tid}} & \overset{\text{def}}{=} \ominus \mathsf{not\_rem}(\texttt{Top}, \texttt{HP}, \texttt{tid}) & \\
& \quad \wedge ((\texttt{HP}+\texttt{tid} \mapsto \_ \ltimes_{\texttt{tid}} \texttt{HP}+\texttt{tid} \mapsto \_) * \mathsf{Id}) & (\text{line } 06) \\
\mathsf{Push}_{\texttt{tid}} & \overset{\text{def}}{=} (\texttt{Top} \mapsto \ell \ltimes_{\texttt{tid}} (\texttt{Top} \mapsto \ell' * \ell' \mapsto \_, \ell)) * \mathsf{Id} & (\text{line 15 in Fig. 1})
\end{array}$$

$$\begin{array}{ll}
G_{\texttt{tid}} & \overset{\text{def}}{=} (\mathsf{Retire}_{\texttt{tid}} \vee \mathsf{Pop}_{\texttt{tid}} \vee \mathsf{Push}_{\texttt{tid}} \vee \mathsf{Reset\_HP}_{\texttt{tid}} \vee \mathsf{Id}) \wedge (I \ltimes I) \\
R_{\texttt{tid}} & \overset{\text{def}}{=} \bigvee_{\texttt{tid'} \in [1..\texttt{th\_num}] \wedge \texttt{tid} \neq \texttt{tid'}} G_{\texttt{tid'}}
\end{array}$$

Fig. 10. Transitions over Shared Resources, and R-G Specifications

The call to `retireNode` is crucial. As we will show below, it ensures that a confirmed hazard pointer cannot be a dangling pointer, and a popped node pointed by any confirmed hazard pointers cannot show up on the stack again (thus the ABA problem is avoided).

Verification of the Algorithm. We first define in Fig. 10 all the operations over the shared data structure, and show which line of the code makes the corresponding transition (read-only operations are simply Id transitions and are omitted). $\mathsf{Pop}_{\texttt{tid}}$ pops the top node from the stack. It requires that the hazard pointer point to the top of the stack. $\mathsf{Retire}_{\texttt{tid}}$ sets the value of $\texttt{HP}[\texttt{tid}]$ into `null`, knowing that the popped node is no longer a hazard node. Then the node ℓ is converted logically from *shared* resource to *private*. $\mathsf{Reset\_HP}_{\texttt{tid}}$ resets the hazard pointer when the thread `tid` fails to pop a node. $\mathsf{Push}_{\texttt{tid}}$ pushes a *private node* onto the stack.

We also define the rely ($R_{\texttt{tid}}$) and guarantee ($G_{\texttt{tid}}$) of the thread `tid`. Here I (defined in (6)) is used to fence the domain of all possible actions. It is easy to see $\mathsf{fence}(I, R_{\texttt{tid}})$ and $\mathsf{fence}(I, G_{\texttt{tid}})$ are satisfied. Next we show some key trace invariants derivable from $\boxminus(R_{\texttt{tid}} \vee G_{\texttt{tid}})$. They are used when the INV rule is applied. Also they show the key intuition of the algorithm.

Invariant 1. This invariant ensures that a node pointed by a hazard pointer is either on the stack or in the set O, so it is safe to dereference a hazard pointer.

$$\begin{array}{l}
\forall \ell, \texttt{tid}, A, O.\ \mathsf{ishazard}(\ell, \texttt{Top}, \texttt{HP}, \texttt{tid}) \wedge \mathsf{opset}(\texttt{Top}, \texttt{HP}, [1..\texttt{th\_num}], O) \\
\qquad\qquad \wedge(\mathsf{stack}(\texttt{Top}, A) * \mathsf{true}) \wedge \ell \neq \texttt{null} \Rightarrow \ell \in A \vee \ell \in O
\end{array}$$

Invariant 2. If a thread `tid` once held a hazard pointer pointing to the top of the stack, and the top node on the stack was popped by other threads, then the node will not be on the stack again as long as `tid`'s `HP` entry is not changed. This invariant ensures that there are no ABA problems.

$$\begin{array}{l}
\forall \ell, A, A', \texttt{tid}. \\
\quad ((\mathsf{ishazard}(\ell, \texttt{Top}, \texttt{HP}, \texttt{tid}) \wedge (\mathsf{stack}(\texttt{Top}, A) * \mathsf{true})) \rhd \texttt{HP}+\texttt{tid} \rightsquigarrow \ell) \\
\qquad\qquad \wedge(\mathsf{stack}(\texttt{Top}, A') * \mathsf{true}) \wedge \ell \neq \texttt{null} \wedge \ell \notin A \Rightarrow \ell \notin A'
\end{array}$$

```
$\mathsf{POP}_{\mathtt{tid}}(\ell, n, \ell') \overset{\text{def}}{=} \ominus((\mathtt{Top} \rightsquigarrow \ell) * (\ell \mapsto n, \ell') * \mathsf{List}(\ell', \_, \_))$
                                  $\wedge\, ((\mathtt{Top} \mapsto \ell \ltimes_{\mathtt{tid}} \mathtt{Top} \mapsto \ell') * \mathsf{Id})$
pop(){
$\{\boxed{\mathtt{HP}+\mathtt{tid} \mapsto \mathtt{null}}_I\}$
01  local done, next, t, t1;
02  done := false;
$\{\boxed{\mathtt{HP}+\mathtt{tid} \mapsto \mathtt{null}}_I\} \wedge \neg\mathtt{done}\}$
$\{\boxed{\mathsf{not\_rem}(\mathtt{Top}, \mathtt{HP}, \mathtt{tid})}_I \wedge \neg\mathtt{done}\}$
loop invariant:
$\{\boxed{\mathsf{not\_rem}(\mathtt{Top}, \mathtt{HP}, \mathtt{tid})}_I \wedge \neg\mathtt{done}$
    $\vee \boxed{\exists n, \ell'.\, \diamondminus\, \mathsf{POP}_{\mathtt{tid}}(\mathtt{t}, n, \ell') \wedge (\mathsf{remove}(\mathtt{t}, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}) * (\mathtt{t} \mapsto n, \ell'))}_I \wedge \mathtt{done}\}$
03  while (!done){
$\{\boxed{\mathsf{not\_rem}(\mathtt{Top}, \mathtt{HP}, \mathtt{tid})}_I\}$
04     <t := [Top]>;
05     if (t == null) return null;
06     <HP[tid] := t>;
$\{\exists \ell.\, \boxed{\mathsf{not\_rem}(\mathtt{Top}, \mathtt{HP}, \mathtt{tid}) \wedge \mathtt{HP}+\mathtt{tid} \mapsto \ell}_I \wedge \mathtt{t} = \ell \wedge \ell \neq \mathtt{null}\}$
07     <t1 := [Top]>;      Apply ATOM-R and ATOM
$\{\exists \ell, \ell'.\, \boxed{\mathsf{not\_rem}(\mathtt{Top}, \mathtt{HP}, \mathtt{tid}) \wedge (\mathtt{HP}+\mathtt{tid} \mapsto \ell * \mathtt{Top} \mapsto \ell')}_I \wedge \mathtt{t} = \ell \wedge \ell \neq \mathtt{null} \wedge \mathtt{t1} = \ell'\}$
$\{\exists \ell, \ell'.\, \boxed{\ell = \ell' \Rightarrow \mathsf{ishazard}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid})}_I \wedge \mathtt{t} = \ell \wedge \ell \neq \mathtt{null} \wedge \mathtt{t1} = \ell'\}$
08     if (t == t1){
$\{\exists \ell.\, \boxed{\mathsf{ishazard}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid})}_I \wedge \mathtt{t} = \ell \wedge \ell \neq \mathtt{null}\}$
                    Apply INV with Invariant 1
$\{\exists \ell, n, \ell'.\, \boxed{\mathsf{ishazard}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}) * \ell \mapsto n, \ell'}_I \wedge \mathtt{t} = \ell\}$
09        <next := t.Next>;
$\{\exists \ell, n, \ell'.\, \boxed{\mathsf{ishazard}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}) * \ell \mapsto n, \ell'}_I \wedge \mathtt{t} = \ell \wedge \mathtt{next} = \ell'\}$
10        <done := CAS(&Top, t, next)>;      Apply INV with Invariant 2
11     }
12  }
$\{\boxed{\exists n, \ell'.\, \diamondminus\, \mathsf{POP}_{\mathtt{tid}}(\mathtt{t}, n, \ell') \wedge (\mathsf{remove}(\mathtt{t}, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}) * (\mathtt{t} \mapsto n, \ell'))}_I\}$
13  retireNode(t);      Apply FRAME and FRAMET
$\{\boxed{\exists n, \ell'.\, \diamondminus\, \mathsf{POP}_{\mathtt{tid}}(\mathtt{t}, n, \ell') \wedge (\mathsf{hazfree}(\mathtt{t}, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}) * (\mathtt{t} \mapsto n, \ell'))}_I\}$
14  <HP[tid] := null>;      Apply ATOM-R and ATOM
$\{\exists n, \ell'.\, \boxed{\diamondminus\, \mathsf{POP}_{\mathtt{tid}}(\mathtt{t}, n, \ell')}_I * (\mathtt{t} \mapsto n, \ell')\}$
15  return t;
$\{\boxed{\diamondminus\, \mathsf{List}(\mathtt{null}, \mathsf{nil})}_I \wedge \mathtt{t} = \mathtt{null} \vee \exists n, \ell'.\, \boxed{\diamondminus\, \mathsf{POP}_{\mathtt{tid}}(\mathtt{t}, n, \ell')}_I * (\mathtt{t} \mapsto n, \ell')\}$
}
```

Fig. 11. Verification of pop

$$I_{\mathrm{thp}} \overset{\mathrm{def}}{=} (\mathtt{Top} \mapsto \_) * I_{\mathrm{hp}}(\mathtt{HP}) \qquad I_{\mathrm{rN}} \overset{\mathrm{def}}{=} \mathsf{remove}(\mathtt{t}, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}) \wedge I_{\mathrm{thp}}$$

$$\mathsf{selfornothazard}(\mathtt{i}) \overset{\mathrm{def}}{=} \boxed{\forall \mathtt{tid}' \in [1..\mathtt{i}-1].\, \mathtt{tid}' = \mathtt{tid} \vee \neg \mathsf{ishazard}(\mathtt{t}, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}')}_{\,I_{\mathrm{rN}}}$$

```
retireNode(t){
{$I_{\mathrm{rN}}$}
  local i, t';
  i := 1;
loop invariant:  {$\mathtt{i} \le \mathtt{th\_num+1} \wedge \mathsf{selfornothazard}(\mathtt{i-1})$}
  while(i<=th_num){
    if (i != tid){
      <t' := HP[i]>;
      if (t '!=  t) {
{$\Diamond\, (\exists \ell. \mathtt{HP}+\mathtt{i} \mapsto \ell \wedge (\mathtt{i} \neq \mathtt{tid} \wedge \ell \neq \mathtt{t})) \wedge \mathsf{selfornothazard}(\mathtt{i-1}) \wedge \mathtt{i} \le \mathtt{th\_num}$}
          Apply INV with Invariant 3
{$\mathsf{selfornothazard}(\mathtt{i}) \wedge \mathtt{i} \le \mathtt{th\_num}$}
          i := i+1;
        }
      }else i:= i+1;
  }
{$\mathsf{selfornothazard}(\mathtt{th\_num+1})$}
{$\boxed{\mathsf{hazfree}(\mathtt{t}, \mathtt{Top}, \mathtt{HP}, \mathtt{tid})}_{\,I_{\mathrm{thp}}}$}
}
```

Fig. 12. Verification of `retireNode`

Invariant 3. This invariant justifies the `retireNode` procedure. If the thread `tid` popped a node ℓ and its `HP` entry points to the node, then for all other thread `tid'` its hazard pointer cannot point to the node and becomes a confirmed hazard pointer again if it was set to point to a different node (ℓ') in history.

$$\forall \ell, \ell', \mathtt{tid}, \mathtt{tid}'.\quad \left(\begin{array}{c} \mathsf{remove}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}) \wedge (\mathtt{HP}+\mathtt{tid}' \rightsquigarrow \ell') \wedge \ell' \neq \ell \\ \triangleright\ \mathtt{HP}+\mathtt{tid} \rightsquigarrow \ell \end{array} \right) \Rightarrow \neg \mathsf{ishazard}(\ell, \mathtt{Top}, \mathtt{HP}, \mathtt{tid}')$$

We show the proof sketch for `pop` in Fig. 11. Here $\boxed{p}_{\,I}$ is used as a shorthand for $(p * \mathsf{true}) \wedge I$. The precondition of `pop` requires that the invariant I hold over the shared resources, and that the calling thread's `HP` entry be initialized to `null`. The post-condition says I holds at the end; the stack was either empty or the node `t` was popped out of the stack and is now part of the local resource of the calling thread. The proof sketch for `retireNode` is given in Fig. 12.

Most part of the proof simply follows the rules of the logic. The interesting part is that the specification of `retireNode` does not mention the linked list and the nodes in opset. Neither does it need to know that the pop operation has been done $(\exists n, \ell'. \Diamond\, \mathsf{POP}_{\mathtt{tid}}(\mathtt{t}, n, \ell'))$. The knowledge can be added back by applying the FRAME and FRAMET rules respectively before we compose `retireNode` with `pop` (see Fig. 11). The `push` procedure has nothing to do with hazard pointers, thus the proof is trivial and can be seen in the extended TR [4].

6 Related Work and Conclusions

Assume-Guarantee (A-G) reasoning [12, 16] often views a concurrent program as an "invariant maintainer" instead of a "predicate transformer" [9], especially for safety verification. With this view, sequential composition of programs seems always trivial and is rarely discussed in previous work on A-G reasoning.

R-G reasoning [8], on the other hand, decomposes specifications into program invariants (R and G) and Hoare-style pre- and post-conditions, which gives us a "predicate transformer" view of programs. With this view, sequential composition of programs ($C_1; C_2$) is no longer trivial. For instance, to use the post condition q of C_1 as the pre-condition of C_2 as in Hoare Logic, we need to ensure the stability of q. Our logic is an extension of R-G reasoning. We take this "predicate transformer" view of programs, and try to spell out the details of the verification step associated with each program construct. Also, our logic successfully combines separation logic and temporal reasoning, which gives us better modularity. The two different frame rules in our logic reflect the frame properties over space (program states) and time respectively.

Gotsman et al. [5] introduced temporal operators in RGSep [19] to reason about liveness properties of non-blocking algorithms. They do not use any past tense operators. Their temporal operators were only used in their rely and guarantee conditions, but not in the pre- and post-conditions. Since the frame rule over R and G was not used there, the interoperability between temporal operators and separation logic operators was not discussed.

Parkinson et al. [14] used CSL to verify safety of the same stack algorithm we verified here. The specifications makes heavy uses of history variables. We believe that our specifications reflect the intuition of the algorithm more directly. Vafeiadis [18] applied RGSep to verify several non-blocking stack algorithms, which all rely on garbage collectors to avoid the ABA problem. It is unclear how the specification of Michael's stacks would look like in RGSep.

Conclusion. In this paper we have proposed a new program logic HLRG, which combines R-G reasoning with past tense temporal assertions to reason about optimistic concurrency algorithms. Our new logic supports modular verification, including the frame rules over both the separation logic and temporal operators. We have verified Michael's lock-free stack with hazard pointers and show that our history logic can directly capture the high-level intuition about the algorithm.

Acknowledgments. We thank anonymous referees for their comments on this paper. Ming Fu and Yong Li are supported in part by China Scholarship Council and by National Natural Science Foundation of China under grant No. 90718026. Much of this work was done during their visits to Yale University in 2009-2010. Zhong Shao is supported in part by NSF grants CNS-0915888, CNS-0910670, and CCF-0811665. Xinyu Feng is supported in part by National Natural Science Foundation of China under grant No. 90818019.

References

[1] Dice, D., Shalev, O., Shavit, N.: Transactional locking II. In: Dolev, S. (ed.) DISC 2006. LNCS, vol. 4167, pp. 194–208. Springer, Heidelberg (2006)

[2] Feng, X.: Local rely-guarantee reasoning. In: Proc. 36th ACM Symp. on Principles of Prog. Lang., pp. 315–327. ACM Press, New York (January 2009)

[3] Feng, X., Ferreira, R., Shao, Z.: On the relationship between concurrent separation logic and assume-guarantee reasoning. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 173–188. Springer, Heidelberg (2007)

[4] Fu, M., Li, Y., Feng, X., Shao, Z., Zhang, Y.: Reasoning about optimistic concurrency using a program logic for history. Technical Report YALEU/DCS/TR-1428, Dept. of Computer Science, Yale University, New Haven, CT (June 2010), http://flint.cs.yale.edu/publications/roch.html

[5] Gotsman, A., Cook, B., Parkinson, M.J., Vafeiadis, V.: Proving that non-blocking algorithms don't block. In: Proc. 36th ACM Symp. on Principles of Prog. Lang., pp. 16–28. ACM, New York (2009)

[6] Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. 13(1), 124–149 (1991)

[7] Herlihy, M., Moss, J.E.B.: Transactional memory: Architectural support for lock-free data structures. In: Proc. 20th Annual Int'l Symp. on Computer Architecture (ISCA), pp. 289–300 (1993)

[8] Jones, C.B.: Tentative steps toward a development method for interfering programs. ACM Trans. on Programming Languages and Systems 5(4), 596–619 (1983)

[9] Lamport, L., Schneider, F.B.: The "Hoare Logic" of CSP, and all that. ACM Trans. Program. Lang. Syst. 6(2), 281–296 (1984)

[10] Lichtenstein, O., Pnueli, A., Zuck, L.D.: The glory of the past. In: Parikh, R. (ed.) Logic of Programs 1985. LNCS, vol. 193, pp. 196–218. Springer, Heidelberg (1985)

[11] Michael, M.M.: Hazard pointers: Safe memory reclamation for lock-free objects. IEEE Trans. Parallel Distrib. Syst. 15(6), 491–504 (2004)

[12] Misra, J., Chandy, K.M.: Proofs of networks of processes. IEEE Trans. Software Eng. 7(4), 417–426 (1981)

[13] 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)

[14] Parkinson, M., Bornat, R., O'Hearn, P.: Modular verification of a non-blocking stack. In: Proc. 34th ACM Symp. on Principles of Prog. Lang, pp. 297–302. ACM Press, New York (January 2007)

[15] Parkinson, M.J., Bornat, R., Calcagno, C.: Variables as resource in hoare logics. In: Proc. 21st Annual IEEE Symposium on Logic in Computer Science (LICS'06), pp. 137–146. IEEE Computer Society, Los Alamitos (August 2006)

[16] Pnueli, A.: In transition from global to modular temporal resoning about programs. In: Apt, K.R. (ed.) Logics and Models of Concurrent Systems. NATO ASI Series, pp. 123–144. Springer, Heidelberg (1984)

[17] Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: Proc. 17th Annual IEEE Symposium on Logic in Computer Science (LICS 2002), pp. 55–74. IEEE Computer Society, Los Alamitos (July 2002)

[18] Vafeiadis, V.: Modular fine-grained concurrency verification. PhD thesis, Computer Laboratory. University of Cambridge, Cambridge, UK (July 2007)

[19] Vafeiadis, V., Parkinson, M.: A marriage of rely/guarantee and separation logic. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 256–271. Springer, Heidelberg (2007)

Theory by Process

Yuxi Fu

BASICS, Department of Computer Science
Shanghai Jiaotong University, Shanghai 200240
MOE-MS Key Laboratory for Intelligent Computing and Intelligent Systems

Abstract. Theories defined in a process model are formalized and studied. A theory in a process calculus is a set of perpetually available processes with finite interactability, each can be regarded as a service, an agent behind the scene or an axiom. The operational and observational semantics of the theories are investigated. The power of the approach is demonstrated by interpreting the asynchronous π-calculus as a theory, the asynchronous theory, in the π-calculus. A complete axiomatic system is constructed for the asynchronous theory, which gives rise to a proof system for the weak asynchronous bisimilarity of the asynchronous π.

1 Introduction

In a network computing environment, how do we evaluate the capacities of two service providers S_1 and S_2? Suppose we place a request to both the providers. One, say S_1, finds the appropriate service in its repertoire and immediately delivers the service. The other provider S_2 does not specialize in the kind of service we are interested in. But since it has a strong search ability, it simply redirects our request to a third party who can offer the service we want for free. Moreover S_2 does it in such a manner that we are not aware of the existence of the third party at all. If S_1 and S_2 can always supply the services with similar qualities, we are led to believe that they are equally powerful. The point is that in a distributed computing environment, we are not testing S_1 and S_2 in isolation. That is clearly impossible. No one can stop S_1 and S_2 from exploring the resources freely available on the network. Although we can pretend that we are testing S_1, what we are really doing is to test S_1 plus all the resources accessible by S_1. Worse still S_1 is often blurred with the environment so that we are not always able to tell precisely which is which. The service providers S_1 and S_2 could have quite different capacities when isolated. They can be however equivalent in a resource rich network environment.

We would like to formalize the above scenario in interaction models. Our basic idea is to regard the services available on the network as a set $\mathcal{S} = \{S_1, S_2, \ldots\}$ of processes. We shall always assume that each member of the set is well-founded. It's no good to have a service that never delivers anything. A one-shot service is not of much use as a piece of resource. So the actual services distributed over different locations are perceived as the processes $!S_1, !S_2, \ldots$. The finite behavior

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 403–416, 2010.

of a process P sitting among $!S_1, !S_2, \ldots$ can always be inspected by considering the finite action sequences of the shape

$$P \,|\, S_{i_1} \,|\, \ldots \,|\, S_{i_k} \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_n} P'$$

since in a finite number of steps, the process P may only consult a finite number of service providers. If we consider a single step action of P, we only have to focus on the actions of the form $P \,|\, S' \xrightarrow{\lambda} P'$ for $S' \in \mathcal{S}$. Now the crucial point is that the services really stay invisibly in the background. So an action $P \,|\, S' \xrightarrow{\lambda} P'$ would appear to us as the action $P \xrightarrow{\lambda} P'$. This is the starting point for the semantics of a theory.

From a model theoretical point of view, sometimes we need to work with an open system within a closed world. The idea is best explained by the notion of asynchrony. The asynchronous π-calculus [Bou92, HT91a, HT91b, HY95, ACS98] is obtained from the synchronous π-calculus by detaching the output prefixes from any continuations. At first look it appears a bit simplistic since asynchrony can not be a syntactical issue. In reality asynchrony is implemented by extra mechanism which we choose to ignore. In a closed model like π-calculus, the detachment between the prefix and the continuation is the only sensible thing to do to achieve asynchrony without introducing any additional gadgets.

There ought to be an alternative treatment to the asynchrony in π-calculus that does not resort to any syntactical manipulation. An output process $\overline{a}c.P$ may communicate to a background environment. The latter picks up the name c and sends it to an input process in a later time. The background environment consists of a bunch of processes of the form $!a(x).\overline{a}x$. These processes are supposed to be hidden from the users. They form a theory in the above sense.

One may also study (constructive) logics in process models. Abramsky's work [Abr93] and a number of related works have shown how to model a logic in a process calculus. But how about a logic theory, say the Peano theory, defined in a logic? From our perspective, a particular logical theory can be formulated as a theory defined in a process model. An element of the theory codes up an axiom of the logic theory. In the process approach to logic, a proposition is interpreted as a process of the form $!A$. To make use of the axiom, an environment has to interact with a copy of A. When the verification (proof) is complete, the environment has gathered enough evidence for the validity of the proposition.

In a similar fashion to the interpretations of logics, programs in different styles have been translated into the π-model [Wal95, HO95, Mil92, CF10]. What is lacking in this research field is any idea about implementations of programming languages. Again the notion of theory is in sight. The implementation of a programming language amounts to defining a theory that codes up the system functions or routines that come with the definition of the language.

The motivation for this paper is that the notion of theory defined by processes arises naturally in many applications of process models. It is worthwhile to give an application independent study of the problems pertaining to such theories. Formal studies in this area are likely to shed new light on some familiar topics.

2 Pi Calculus

The π-calculus of Milner, Parrow and Walker [MPW92] has been widely studied in both theory and in practice. Our presentation of the model is slightly different from the standard one. The main difference is that we draw a firm line between the names and the name variables. The reader is advised to consult [FZ10] for the discussion why the distinction between the two categories is important. Throughout the paper the following notational convention will be observed:

- The set $\mathcal{N}$ of the *names* is ranged over by a, b, c, d, e, f, g, h.
- The set $\mathcal{N}_v$ of the *name variables* is ranged over by u, v, w, x, y, z.
- The set $\mathcal{N} \cup \mathcal{N}_v$ is ranged over by l, m, n, o, p, q.

An assignment ρ is a partial function from $\mathcal{N}_v$ to $\mathcal{N}$ whose domain of definition is cofinite. A condition is a finite conjunction of atomic propositions. An atomic proposition is either a match $[m{=}n]$ or a mismatch $[m{\neq}n]$. We always omit the conjunction operator. The set of the conditions is ranged over by ϕ, φ, ψ. We write $\varphi \Leftrightarrow \top$ ($\varphi \Leftrightarrow \bot$) if φ is evaluated to the true value $\top$ (the false value $\bot$) no matter how the name variables appearing in the condition are instantiated. Similarly we can define $\varphi \Rightarrow \psi$ and $\varphi \Leftrightarrow \psi$.

Our definition of the π-calculus is influenced by the results obtained in [Fu10]. It is equivalent to the standard presentation in terms of expressiveness and it has better algebraic property. The set of *terms* is inductively constructed from the following grammar:

$$T := \sum_{i \in I} \varphi_i \lambda_i.T_i \mid T \,|\, T' \mid (c)T \mid !\pi.T,$$

where $\lambda_i \in \{n(x), \overline{n}m, \overline{n}(c)\} \cup \{\tau\}$ and $\pi \in \{n(x), \overline{n}m, \overline{n}(c)\}$; they are prefixes. Here τ indicates an interaction, $ab, \overline{a}b, \overline{a}(c)$ denote respectively an input action, an output action and a bound output action. In the guarded choice $\sum_{i \in I} \varphi_i \lambda_i.T_i$ the indexing set I must be finite. We shall often write $\varphi_1\lambda_1.T_1 + \ldots + \varphi_n\lambda_n.T_n$ for $\sum_{i \in \{1,..,n\}} \varphi_i\lambda_i.T_i$. We write $\mathbf{0}$ for the guarded choice whose indexing set is the empty set. Due to the set theoretical nature the guarded choice $\varphi_1\lambda_1.T_1 + \ldots + \varphi_i\lambda_i.T_i + \varphi_{i+1}\lambda_{i+1}.T_{i+1} + \ldots + \varphi_n\lambda_n.T_n$ is the same as $\varphi_1\lambda_1.T_1 + \ldots + \varphi_{i+1}\lambda_{i+1}.T_{i+1} + \varphi_i\lambda_i.T_i + \ldots + \varphi_n\lambda_n.T_n$. Sometimes we will abuse notation by writing for instance $\varphi_0\lambda_0.T_0 + \sum_{i \in \{1,2\}} \varphi_i\lambda_i.T_i$ for $\varphi_0\lambda_0.T_0 + \varphi_1\lambda_1.T_1 + \varphi_2\lambda_2.T_2$. We will also abbreviate $[\top]\lambda.T + \sum_{i \in I} \varphi_i\lambda_i.T_i$ to $\lambda.T + \sum_{i \in I} \varphi_i\lambda_i.T_i$. A name c appearing underneath the localization operator (c) or a bound output prefix $\overline{a}(c)$ is local. A name is global if it is not local. A name variable x is bound if it is underneath an input prefix, say $n(x)$. A name variable is free if it is not bound. Both local names and bound name variables are subject to α-conversion. We will use the functions $gn(\_), ln(\_), n(\_), fv(\_), bv(\_), v(\_)$ with the obvious meanings. A *process* is a term in which all the name variables are bound. Let $\mathcal{T}$ denote the set of the terms, ranged over by R, S, T, and $\mathcal{P}$ the set of the processes, ranged over by $A, B, C, \ldots, O, P, Q$. The term $!\pi.T$ is in replication form. A term without any occurrence of the replication operator is called *finite*.

The operational semantics of this π-calculus is defined by the labeled transition system generated inductively from the following rules.

Action

$$\frac{}{\sum_{i\in I}\varphi_i\lambda_i.T_i \xrightarrow{ac} T_i\{c/x\}}\ \begin{array}{l}\varphi_i \Leftrightarrow \top,\\ \lambda_i = a(x).\end{array} \qquad \frac{}{\sum_{i\in I}\varphi_i\lambda_i.T_i \xrightarrow{\lambda_i} T_i}\ \begin{array}{l}\varphi_i \Leftrightarrow \top,\\ \lambda_i \text{ is not}\\ \text{an input.}\end{array}$$

Composition

$$\frac{T \xrightarrow{\lambda} T'}{S\,|\,T \xrightarrow{\lambda} S\,|\,T'} \qquad \frac{S \xrightarrow{ab} S' \qquad T \xrightarrow{\overline{a}b} T'}{S\,|\,T \xrightarrow{\tau} S'\,|\,T'} \qquad \frac{S \xrightarrow{ac} S' \qquad T \xrightarrow{\overline{a}(c)} T'}{S\,|\,T \xrightarrow{\tau} (c)(S'\,|\,T')}$$

Localization

$$\frac{T \xrightarrow{\overline{a}c} T'}{(c)T \xrightarrow{\overline{a}(c)} T'} \qquad \frac{T \xrightarrow{\lambda} T'}{(c)T \xrightarrow{\lambda} (c)T'}\ c \notin n(\lambda)$$

Replication

$$\frac{}{!a(x).T \xrightarrow{ab} T\{b/x\}\,|\,!a(x).T} \qquad \frac{}{!\pi.T \xrightarrow{\pi} T\,|\,!\pi.T}\ \text{if } \pi \text{ is not an input.}$$

We have omitted all the symmetric rules. There is a side condition $ln(\lambda) \cap gn(T) = \emptyset$ on the first composition rule. These remarks also apply to the labeled transition systems defined later. Let $\Longrightarrow$ be the reflexive and transitive closure of $\xrightarrow{\tau}$. Let $\overset{\widehat{\lambda}}{\Longrightarrow}$ be $\Longrightarrow$ if $\lambda = \tau$ and $\Longrightarrow\xrightarrow{\lambda}\Longrightarrow$ otherwise. These notations allow us to define Milner and Park's bisimulation equality [Mil89].

Definition 1. *A symmetric relation $\mathcal{R}$ on $\mathcal{P}$ is a* weak bisimulation *if $Q \overset{\widehat{\lambda}}{\Longrightarrow} Q'\mathcal{R}P'$ whenever $Q\mathcal{R}P \xrightarrow{\lambda} P'$. The* weak bisimilarity $\simeq$ *is the largest weak bisimulation.*

We write $S \simeq T$ if $S\rho \simeq T\rho$ for every assignment ρ whose domain of definition is disjoint from $bv(S\,|\,T)$. An example of bisimulation equality is $!\pi.T \simeq \pi.(T\,|\,!\pi.T)$. For the particular π-calculus of this paper, $\simeq$ is closed under all the operators.

Theorem 1. *The relation $\simeq$ is both an equivalence and a congruence on $\mathcal{T}$.*

Theorem 1 relies on the fact that in the guarded choice $\sum_{i\in I}\varphi_i\lambda_i.T_i$ the operator is $\sum_{i\in I}\varphi_i\lambda_i.\_$. From $\tau \simeq [x{=}y]\tau$ we may derive $\overline{a}a + \tau.\tau \simeq \overline{a}a + \tau.[x{=}y]\tau$; but we may not derive $\overline{a}a + \tau \simeq \overline{a}a + [x{=}y]\tau$.

A process P is *well-founded* if there is no infinite action sequence $P \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_i} \ldots$ starting from P. It is a process with *finite interactability* if it is well-founded and there is a number $k \neq 0$ such that no action sequence $P \xrightarrow{\lambda_1} P_1 \ldots \xrightarrow{\lambda_i} P_i$ of P contains more than k non-τ actions. A process P is *functional* if every maximal action sequence of P is of the form $P \xrightarrow{\lambda}\Longrightarrow\xrightarrow{\lambda'} P'$, where λ is an input action and λ' an output action.

3 Theory

Upon request, a service provider should deliver the service in a short time. In a similar token, a proposition should have a finite description so that its validity can be verified in a finite number of steps. These observations lead to the following definition.

Definition 2. *A* theory $\mathbf{A}$ *is a nonempty set of processes of the form* $\pi.T$ *with finite interactability. These processes are called* axioms*.*

The intuition behind a theory is that it provides a set of eternal truths within a closed world. The finite interactability condition ensures that a service must be delivered within an expected number of interactions. Operationally an axiom A can be identified to the process $!A$. Every proof in the closed world can make inquiry into these laws. Propositions valid in the model are all relative to the set of the eternal truths. By definition every axiom A is nontrivial in the sense that $A \not\asymp \mathbf{0}$. A theory $\mathbf{A}$ is *finite* if it is a finite set. It is *finitely presentable* if it can be generated from a finite theory. By $\mathbf{A}$ being generated from $\mathbf{B}$, we mean that

$$\mathbf{A} = \{B\alpha \mid B \in \mathbf{B}, \text{ and } \alpha \text{ is an injective function from } \mathcal{N} \text{ to } \mathcal{N}\}.$$

A theory is *functional* if all its axioms are functional. A theory is *recursive* if it is a recursive set of finite processes.

Let's see some examples.

Example 1. The asynchronous theory **Asy** is defined by the finitely presentable theory $\{a(x).\overline{a}x \mid a \in \mathcal{N}\}$. In the presence of **Asy** communications are asynchronous. An output process $\overline{a}c.P$ does not have to interact with the target process. It could interact with the axiom $a(x).\overline{a}x$ and let the latter pass the information to the target. Using the same idea, one may define the finite theory $\mathbf{AB} = \{a(x).\overline{b}x, b(x).\overline{a}x\}$ that essentially identifies the names a and b.

Example 2. The natural numbers can be coded up in the following fashion:

$$[\![\underline{0}]\!]_p \overset{\text{def}}{=} \overline{p}\bot,$$
$$[\![\underline{i+1}]\!]_p \overset{\text{def}}{=} (q)(\overline{p}q \mid [\![\underline{i}]\!]_q).$$

Here p is the access name for a number. The notation $\bot$ is a name that denotes false; similarly the name $\top$ denotes true. The natural numbers are underlined to avoid any confusion. For simplification the following derived prefix is introduced:

$$\overline{a}(\underline{i}).T \overset{\text{def}}{=} \overline{a}(p).(T \mid [\![\underline{i}]\!]_p). \tag{1}$$

It is routine to define processes $SUCC_a$, ADD_a, MUL_a, EQ_a and L_a that implement the successor function, the addition, the multiplication, the equality predicate and the linear order predicate on the natural numbers. The process L_a for example inputs a local name at a; and then uses that local name to get three more local names, say b, c, d. It continues to input a number $\underline{i}$ at b and a number $\underline{j}$ at c. Finally it returns $\top$ at d if $i < j$; otherwise it returns $\bot$ at d. Let **Pa** be the following theory

$$\{\overline{p_0}(\underline{0}), \ldots, \overline{p_i}(\underline{i}), \ldots\} \cup \{SUCC_a, ADD_a, MUL_a, EQ_a, L_a \mid a \in \mathcal{N}\}.$$

It is an implementation of the Peano arithmetic in π-calculus. For details of the encodings the reader could consult [FZ10].

Example 3. The theory **Crp** is given by the union of **Pa** with the following set

$$\{e(v).v(y).Enc, d(v).v(z).Dec\}.$$

The action sequences of the encryption function $e(v).v(y).Enc$ are all of the form

$$e(v).v(y).Enc \xrightarrow{ec} \xrightarrow{cf} \overset{\tau}{\Longrightarrow} \xrightarrow{\overline{c}(g)} \simeq [\![\underline{i}]\!]_g$$

for some number i. After receiving a local name c, it inputs at c a name that points to a number, the plain text, and then outputs at c a name that points to the number $\underline{i}$, the encrypted text, after completing a sequence of internal computations. The decryption function $d(v).v(z).Dec$ has the dual semantics. The theory **Crp** provides an encryption/decryption facility every process can make use of. A complete specification of **Crp** depends on the choice of the encryption and decryption functions.

Example 4. Suppose that we would like to define a random number generator that provides perpetual service on network. It appears at first sight that the theory **Ran** can be defined by extending **Pa** with the process

$$a(v).(c)(\overline{c}(\underline{0}) \mid !c(x).(\overline{c}(d).\overline{d}x + \overline{v}x)). \tag{2}$$

Upon receiving a private channel provided by a user, the process (2) randomly generates a number and sends it to the user through the private channel. However process (2) may diverge. So it is not an axiom according to our definition. The theory **Ran** can be defined by the finite set $\{\overline{g}(\underline{0}), g(x).\overline{g}(p).\overline{p}x)\}$. It can also be defined by the infinite set $\{\overline{g}(\underline{0}), \overline{g}(\underline{1}), \ldots, \overline{g}(\underline{i}), \ldots\}$. It is worth remarking that the randomness is achieved by the nondeterminism. There is no other way.

Example 5. The π-calculus has been used both as a specification language and a machine language. The rational behind these practices is that π is expressive enough to qualify for a machine model. Now if we think of π-calculus as a machine model, we can talk about programming in π-calculus. This is precisely what is done in [Wal95]. Formally what is then an interpreter of a higher order programming language on π? Whatever the interpreter is, it must give an account of the standard routines and packages supplied by the programming language. In our opinion these routines and packages are best interpreted as a theory **Prg**. Two programs defined according to the grammar of the language are equivalent if they are so in the presence of **Prg**. Let O be a program that invokes a system routine and P be a user defined program that achieves the same functionality. Conceptually O and P are equivalent. But they are not bisimilar since the former may interact at a name which P does not know. The notion of theory is a starting point to address issues of this kind.

3.1 Semantics

To investigate the algebraic properties of the theories, we need to define the operational semantics of the theories first. The power of a theory $\mathbf{A}$ is duly exhibited by the 'process'

$$\prod_{A\in\mathbf{A}} !A,$$

which is not always admissible at the syntactical level since it makes use of a possibly infinite composition. A process P under theory $\mathbf{A}$ can be imagined as a fixpoint in the following sense:

Operationally P is the same as $P \mid \prod_{A\in\mathbf{A}} !A$.

But notice that $P \mid A$, for each $A \in \mathbf{A}$, is also a fixpoint of the same nature. By exploring the fact that $\prod_{A\in\mathbf{A}} !A$ is equivalent to $A \mid \prod_{A\in\mathbf{A}} !A$, one sees that $P \mid A$ is operationally the same as P. The semantics of the theory $\mathbf{A}$, or the semantics of the $\pi_{\mathbf{A}}$-calculus, extends the operational semantics of the π-calculus with the following rule:

$$\frac{P \mid A \xrightarrow{\lambda} P'}{P \xrightarrow{\lambda} P'}\ A \in \mathbf{A}. \tag{3}$$

Definition 1 can be immediately applied to the $\pi_{\mathbf{A}}$-calculus.

Definition 3. *A symmetric relation $\mathcal{R}$ on $\mathcal{P}$ is a* weak $\mathbf{A}$-bisimulation *if $Q \stackrel{\widehat{\lambda}}{\Longrightarrow} Q'\mathcal{R}P'$ in $\pi_{\mathbf{A}}$ whenever $Q\mathcal{R}P \xrightarrow{\lambda} P'$ in $\pi_{\mathbf{A}}$. The* weak $\mathbf{A}$-bisimilarity $\simeq_{\mathbf{A}}$ *is the largest weak $\mathbf{A}$-bisimulation.*

The proof of Theorem 1 can be repeated to show that $\simeq_{\mathbf{A}}$ is both an equivalence and a congruence on $\mathcal{T}$.

Since every axiom in a theory is nontrivial, the fact stated in the next proposition is apparent.

Proposition 1. *The strict inclusion $\simeq \subset \simeq_{\mathbf{A}}$ holds for every theory $\mathbf{A}$.*

Proof. Using the fact that $\simeq$ is closed under composition, it is easy to show that $\simeq$ is an $\mathbf{A}$-bisimulation. The inclusion is strict since $\mathbf{A}$ is nonempty. □

The next lemma is a generalization of Proposition 1.

Lemma 1. *If $\mathbf{A} \subseteq \mathbf{B}$ then $\simeq_{\mathbf{A}} \subseteq \simeq_{\mathbf{B}}$.*

By abusing the notation again, one could describe the relationship between $\mathbf{0}$ and the theory $\mathbf{A}$ by the following statement:

Operationally $\mathbf{0}$ is the same as $\prod_{A\in\mathbf{A}} !A$.

The equivalence has been exploited to define the semantics of the asynchronous π-calculus. Honda and Tokoro introduce the following rule in [HT91a, HT91b].

$$\frac{}{\mathbf{0} \xrightarrow{ac} \overline{a}c} \tag{4}$$

It is evident that (4) is essentially (3) applied to $\mathbf{Asy}$.

3.2 Kernel

A theory $\mathbf{A}$ is *consistent* if $\simeq_{\mathbf{A}}$ is not $\mathcal{P}\times\mathcal{P}$; it is inconsistent otherwise. The next proposition is useful.

Proposition 2. *The following statements are equivalent:*
(i) $\mathbf{A}$ *is consistent;*
(ii) $\exists P{\in}\mathcal{P}.P \not\simeq_{\mathbf{A}} \mathbf{0}$*;*
(iii) $\exists P{\in}\mathcal{P}.\forall A{\in}\mathbf{A}.P \not\simeq_{\mathbf{A}} A$*.*

Proof. If (ii) did not hold, then every process would be equated by theory $\mathbf{A}$, contradicting (i). Hence (i) implies (ii). If $\forall P{\in}\mathcal{P}.\exists A{\in}\mathbf{A}.P \simeq_{\mathbf{A}} A$, then every process is equated to $\mathbf{0}$. So (ii) implies (iii). Finally (iii) trivially implies (i). □

The above proposition indicates that there is a distinguishing line between the processes equal to $\mathbf{0}$ in the equational theory of $\mathbf{A}$ and those that are not. This motivates the following definition: The *kernel* $\mathbf{A}_{ker}$ of the theory $\mathbf{A}$ is the set of the processes equal to $\mathbf{0}$ under the theory $\mathbf{A}$, i.e.

$$\mathbf{A}_{ker} = \{A \mid A \simeq_{\mathbf{A}} \mathbf{0}\}.$$

By Proposition 2, a theory is consistent if and only if its kernel is not $\mathcal{P}$.

Proposition 3. *The* $\mathbf{A}$*-bisimilarity equals the* $\mathbf{B}$*-bisimilarity iff* $\mathbf{A}_{ker} = \mathbf{B}_{ker}$*.*

Proof. It is clear that $\mathbf{A}_{ker} \subseteq \mathbf{B}_{ker}$ if and only if $\simeq_{\mathbf{A}}\subseteq\simeq_{\mathbf{B}}$. □

A corollary of Proposition 3 is that the power of a theory is essentially determined by its kernel. One could define for instance that $\mathbf{A}$ is a subtheory of $\mathbf{B}$ if $\mathbf{A} \subseteq \mathbf{B}_{ker}$, and that A is essentially in $\mathbf{A}$ if $A \in \mathbf{A}_{ker}$.

Although it is easy to see that $\mathbf{Asy}$ is consistent, it is generally a tricky job to establish the consistency of a theory. Let $\mathbf{F}$ be the recursive theory consisting of all the finite processes. For each process P, let $P^{\neg(!)}$ denote the process obtained from P by removing all the occurrences of the replication operator and the localization operator. It is not difficult to see that $P \simeq_{\mathbf{F}} P^{\neg(!)} \simeq_{\mathbf{F}} \mathbf{0}$. So $\mathbf{F}_{ker} = \mathcal{P}$. Therefore $\mathbf{F}$ is inconsistent. For a positive result, we remark that all functional theories are consistent. In a functional theory the process $\overline{a}a$ is never equal to $\mathbf{0}$.

4 Asynchronous Theory and Asynchronous π

We prove in this section that $\mathbf{Asy}$ provides a faithful account of the asynchronous π-calculus. We adopt the following grammar for the asynchronous π-calculus, which summarizes the essential feathers of the calculi defined in literature [Bou92, HT91a, HT91b, HY95, ACS98].

$$T := \mathbf{0} \mid \overline{n}m \mid \sum_{i\in I} n_i(x).T_i \mid T \,|\, T \mid (c)T \mid !n(x).T.$$

Notice that the above grammar maintains a distinction between the names and the name variables. There are basically two ways to formulate the semantics of the asynchronous π-calculus. Honda and Tokoro's semantics makes use of the rule (4). The notion of theory is lurking in their framework. Amadio, Castellani and Sangiorgi's approach takes a more traditional view on the asynchronous π. Their operational semantics is defined by the following rules.

Action

$$\frac{}{\overline{a}b \xrightarrow{\overline{a}b} \mathbf{0}} \qquad \frac{}{\sum_{i\in I} a_i(x).T_i \xrightarrow{a_i c} T_i\{c/x\}}$$

Composition

$$\frac{T \xrightarrow{\lambda} T'}{S \,|\, T \xrightarrow{\lambda} S \,|\, T'} \qquad \frac{S \xrightarrow{ab} S' \qquad T \xrightarrow{\overline{a}b} T'}{S \,|\, T \xrightarrow{\tau} S' \,|\, T'} \qquad \frac{S \xrightarrow{ac} S' \qquad T \xrightarrow{\overline{a}(c)} T'}{S \,|\, T \xrightarrow{\tau} (c)(S' \,|\, T')}$$

Localization

$$\frac{T \xrightarrow{\overline{a}c} T'}{(c)T \xrightarrow{\overline{a}(c)} T'} \qquad \frac{T \xrightarrow{\lambda} T'}{(c)T \xrightarrow{\lambda} (c)T'}\ c \notin n(\lambda)$$

Replication

$$\frac{}{!a(x).T \xrightarrow{ac} T\{c/x\} \,|\, !a(x).T}$$

In Honda and Tokoro's treatment the asynchronous π differs from the synchronous π at the operational level, whereas in Amadio, Castellani and Sangiorgi's approach it is at the observational level. The definition of the asynchronous bisimilarity [ACS98] appears odd from the point of view of interaction.

Definition 4. *A symmetric relation $\mathcal{R}$ on the asynchronous π-processes is an* asynchronous bisimulation *if the following statements are valid whenever $P\mathcal{R}Q$:*

1. *If $Q \xrightarrow{\tau} Q'$ then $P \Longrightarrow P'\mathcal{R}Q'$ for some P'.*
2. *If $Q \xrightarrow{\overline{a}b} Q'$ then $P \overset{\overline{a}b}{\Longrightarrow} P'\mathcal{R}Q'$ for some P'.*
3. *If $Q \xrightarrow{\overline{a}(b)} Q'$ then $P \overset{\overline{a}(b)}{\Longrightarrow} P'\mathcal{R}Q'$ for some P'.*
4. *If $Q \xrightarrow{ab} Q'$ then either $P \overset{ab}{\Longrightarrow} P'\mathcal{R}Q'$ for some P' or $P \Longrightarrow P'$ for some P' such that $P' \,|\, \overline{a}b \,\mathcal{R}\, Q'$.*

The asynchronous bisimilarity $\simeq_a$ is the largest asynchronous bisimulation.

The asynchronous π is a syntactic subcalculus of π. It is also an operational variant of π according to Honda and Tokoro's formulation. Amadio, Castellani and Sangiorgi have proved that $\simeq_a$ coincides with Honda and Tokoro's bisimulation equivalence, called HT-bisimilarity in [ACS98]. Their proof can be extended to produce a proof of the following theorem.

Theorem 2. *Let S, T be asynchronous π-terms. Then $S \simeq_a T$ iff $S \simeq_{\mathbf{Asy}} T$.*

Theorem 2 can be interpreted as saying that the asynchronous π is a syntactical simplification of $\pi_{\mathbf{Asy}}$. It perceives $\pi_{\mathbf{Asy}}$ as a submodel of the π-calculus. It can also be seen as a justification of the asynchronous π as defined by Honda and Tokoro, as well as the variant defined by Amadio, Castellani and Sangiorgi.

L1	$(c)\mathbf{0} = \mathbf{0}$	
L2	$(c)(d)T = (d)(c)T$	
L3	$(c)([x{=}c]\varphi\lambda.T + \sum) = (c)\sum$	
L4	$(c)([x{\neq}c]\varphi\lambda.T + \sum) = (c)(\varphi\lambda.T + \sum)$	
L5	$(c)(\varphi\lambda.T + \sum) = (c)\sum$	if $\exists d \in \mathcal{N}.\lambda = \overline{c}d \vee \lambda = \overline{c}(d)$
L6	$(c)(\varphi\overline{n}c.T + \sum) = (c)(\varphi\overline{n}(c).T + \sum)$	if $c \notin gn(\varphi) \wedge c \neq n$
L7	$(c)\sum_{i\in I}\varphi_i\lambda_i.T_i = \sum_{i\in I}\varphi_i\lambda_i.(c)T_i$	if $\forall i \in I.c \notin gn(\varphi_i, \lambda_i)$
M1	$[\bot]\lambda.T + \sum = \sum$	
M2	$\varphi\lambda.T + \sum = \psi\lambda.T + \sum$	if $\varphi \Leftrightarrow \psi$
M3	$[x{=}p]\varphi\lambda.T + \sum = [x{=}p](\varphi\lambda.T)\{p/x\} + \sum$	
M4	$[x{\neq}p]\varphi\lambda.\sum' + \sum = [x{\neq}p]\varphi\lambda.[x{\neq}p]\sum' + \sum$	
S1	$\varphi\lambda.T + \sum = \varphi\lambda.T + \varphi\lambda.T + \sum$	
S2	$\varphi\lambda.T + \sum = [x{=}p]\varphi\lambda.T + [x{\neq}p]\varphi\lambda.T + \sum$	
S3	$\varphi n(x).S + \varphi n(x).T + \sum = \varphi n(x).S + \varphi n(x).T + \varphi n(x).([x{=}p]\tau.S + [x{\neq}p]\tau.T) + \sum$	
T1	$\varphi\tau.\sum = \varphi\sum$	
T2	$\sum + \varphi\tau.\sum = \sum$	
T3	$\phi\lambda.(\varphi\tau.T + \sum) + \sum' = \phi\lambda.(\varphi\tau.T + \sum) + \phi\varphi\lambda.T + \sum'$	

Fig. 1. Axioms for the Weak Bisimilarity

5 Proof System

A complete equational system for the strong asynchronous bisimilarity is given in [ACS98]. Such a system for the weak asynchronous bisimilarity has not been available. The problem in generalizing a result from the strong case to the weak case could be an indication that something is not quite right. The difficulty in designing an equational system for the weak asynchronous bisimilarity is due to the lack of the output prefix operator. This is unfortunate since the role of the output prefix operation is to impose orders on interactions. Its relationship to asynchrony, or synchrony for that matter, was not intended. Our approach disowns this problem.

The expansion law plays a crucial role in proof systems. It is about how to convert two concurrent choice terms to one choice term. Suppose S, T are respectively the guarded choices $\sum_{i\in I}\varphi_i\lambda_i.S_i$ and $\sum_{j\in J}\psi_j\lambda_j.T_j$. Then

$$\begin{aligned} S \mid T = {} & \sum_{i\in I}\varphi_i\lambda_i.(S_i \mid T) + \sum_{i\in I, j\in J}^{\lambda_i=m(x),\lambda_j=\overline{n}p} \varphi_i\psi_j[m{=}n]\tau.(S_i\{p/x\} \mid T_j) \\ & + \sum_{j\in J}\psi_j\lambda_j.(S \mid T_j) + \sum_{i\in I, j\in J}^{\lambda_j=m(x),\lambda_i=\overline{n}p} \varphi_i\psi_j[m{=}n]\tau.(S_i \mid T_j\{p/x\}). \end{aligned}$$

Let AS be the equational system defined in Figure 1 plus the expansion law. In Figure 1 the notation $\sum$ stands for $\sum_{i\in I}\varphi_i\lambda_i.T_i$ and $\sum'$ for $\sum_{j\in J}\psi_j\lambda_j.T_j$. Accordingly $\varphi\sum$ should be understood as $\sum_{i\in I}\varphi\varphi_i\lambda_i.T_i$. Our axiomatic system differs from the standard one in that it is defined in terms of the guarded choice operator rather than the unguarded choice operator.

In AS we may rewrite terms to normal forms, whose definition is given next.

Definition 5. *Let $\mathcal{F}$ be $gn(T) \cup fv(T)$. A finite π-term T is a* normal form *on $\mathcal{F}$ if $T \equiv \sum_{i \in I} \varphi_i \lambda_i.T_i$ such that for each $i \in I$ one of the followings holds.*

1. *If $\lambda_i = \tau$ then T_i is a normal form on $\mathcal{F}$.*
2. *If $\lambda_i = \overline{n}m$ then T_i is a normal form on $\mathcal{F}$.*
3. *If $\lambda_i = \overline{n}(c)$ then $T_i \equiv [c \notin \mathcal{F}]T_i^c$ for some normal form T_i^c on $\mathcal{F} \cup \{c\}$.*
4. *If $\lambda_i = n(x)$ then T_i is of the form*
$$[x \notin \mathcal{F}]T_i^{\neq} + \sum_{m \in \mathcal{F}} [x{=}m]T_i^m$$
such that $T_i^{\neq}$ is a normal form on $\mathcal{F} \cup \{x\}$ and, for each $m \in \mathcal{F}$, $x \notin fv(T_i^m)$ and T_i^m is a normal form on $\mathcal{F}$.

For the motivation of the above definition and the proof of the next lemma, the reader is referred to [FZ10].

Lemma 2. *If T is finite, then a normal form T' exists such that $AS \vdash T = T'$.*

AS is sound and complete for the weak bisimilarity on the finite π-terms.

Theorem 3. *Suppose S, T are finite. Then $S \simeq T$ iff $AS \vdash S = T$.*

Complete systems have been discussed in literature [MPW92, PS95, Lin95, FY03]. A recent account that fits more into the present context can be found in [FZ10]. Notice that our formulation of T2 is crucial for the completeness proof.

We now turn to $\simeq_{\mathbf{Asy}}$. Let $AS_{\mathbf{Asy}}$ be AS together with the following law

$$a(x).\overline{a}x = \mathbf{0}. \tag{5}$$

Apparently $AS_{\mathbf{Asy}}$ is sound for $\simeq_{\mathbf{Asy}}$. The first indication that (5) is complete is the validity of the saturation property.

Lemma 3 (saturation). *Suppose that T is a normal form. The following statements are valid in the $\pi_{\mathbf{Asy}}$-calculus for every assignment ρ whose domain of definition is disjoint from $bv(T)$.*

1. *If $T\rho \stackrel{\lambda}{\Longrightarrow} T'$ and λ is not an input action, then $AS_{\mathbf{Asy}} \vdash T = T + \varphi\lambda'.T'$ for some φ, λ' such that $\varphi\rho \Leftrightarrow \top$ and $\lambda'\rho = \lambda$.*
2. *If $T\rho \stackrel{ae}{\Longrightarrow} T'$, $e \notin n(T\rho)$ and $z \notin v(t)$, then $AS_{\mathbf{Asy}} \vdash T = T + \varphi n(z).T'\{z/e\}$ for some φ, n such that $\varphi\rho \Leftrightarrow \top$ and $\rho(n) = a$.*

Proof. If $T \stackrel{\lambda}{\Longrightarrow} T'$ makes use of the rule (3) k times, then it is easy to see that $T \,|\, a_1(x).\overline{a_1}x \,|\, \ldots \,|\, a_k(x).\overline{a_k}x \stackrel{\lambda}{\Longrightarrow} T'$. By Lemma 2 there is some normal form T_1 such that $AS \vdash T_1 = T \,|\, a_1(x).\overline{a_1}x \,|\, \ldots \,|\, a_k(x).\overline{a_k}x$. By the standard approach it is easy to establish that $AS_{\mathbf{Asy}} \vdash T_1 = T_1 + \lambda.T'$. Thus $AS_{\mathbf{Asy}} \vdash T = T \,|\, a_1(x).\overline{a_1}x \,|\, \ldots \,|\, a_k(x).\overline{a_k}x = T_1 = T_1 + \lambda.T' = T + \lambda.T'$. Notice that according to (5) the equality $T = T \,|\, a(x).\overline{a}x$ follows from $T = T \,|\, \mathbf{0}$, which in turn follows from the expansion law.

This simple argument should be enough for an informed reader. □

The proof of the completeness theorem is an induction on the complexity of the normal forms. The depth $dep(T)$ of a normal form T is defined as follows:

$$\begin{aligned} dep(\mathbf{0}) &\overset{\text{def}}{=} 0, \\ dep(\varphi n(x).T) &\overset{\text{def}}{=} dep(T)+2, \\ dep(\varphi\lambda.T) &\overset{\text{def}}{=} dep(T)+1, \text{ if } \lambda \text{ is not an input}, \\ dep(\sum_{i\in I}\varphi_i\lambda_i.T_i) &\overset{\text{def}}{=} \max\{dep(\varphi_i\lambda_i.T_i)\}_{i\in I}. \end{aligned}$$

For a finite term T, $dep(T)$ is defined by $dep(T')$, where T' is a normal form of T. It is worth remarking that the depth for input prefix is greater than that for output, bounded output and tau prefixes. This is important to the next proof.

Theorem 4 (completeness). *For finite S, T, $S \simeq_{\mathbf{Asy}} T$ iff $AS_{\mathbf{Asy}} \vdash S = T$.*

Proof. Suppose $P \equiv \sum_{i\in I}\varphi_i\lambda_i.S_i$ and $Q \equiv \sum_{j\in J}\psi_j\lambda_j.T_j$ are normal forms such that $P \simeq_{\mathbf{Asy}} Q$. If $a(x).S_i$ is a summand of P, then $P \xrightarrow{ac} S_i\{c/x\}$. The process Q has to simulate this action in the following manner $Q \Longrightarrow Q_1 \xrightarrow{ac} Q_2 \Longrightarrow Q'$. It is easy to see that $dep(Q_1) \leq dep(Q)$ and $dep(Q') \leq dep(Q_2)$. If $Q_1 \xrightarrow{ac} Q_2$ is not derived from the rule (3), then clearly $dep(Q_2) < dep(Q_1)$. If it is derived from the rule (3), then $AS \vdash Q_2 = Q_2'$ for some normal form Q_2', using the expansion law. It is obvious that $dep(Q_2) = dep(Q_2') \leq dep(Q_1)+1$ by the definition of the depth function. But $dep(S_i\{c/x\}) \leq dep(P)-2$ by definition. Hence $dep(Q') + dep(S_i\{c/x\}) < dep(Q) + dep(P)$. So $AS_{\mathbf{Asy}} \vdash Q' = S_i\{c/x\}$ by induction hypothesis.

The above oversimplified account is meant to bring out the fact that the depth function $dep(\_)$ does allow the standard inductive proof to go through. Consult [FZ10] for the general idea of the completeness proof. □

6 Conclusion

The notion of theory probably should have been introduced long time ago. A theory is essentially an implementation. In practice it is one of the most important concepts to start with. Theories can be defined for process calculi other than the π-calculus. The most interesting theories are defined in complete process calculi [FY03]. In a complete model the Peano arithmetic can be defined in a robust manner. Example 2 through Example 5 of Section 3 would not make a lot of sense for incomplete models like CCS (the incompleteness of CCS is established in [FY03]).

There are several directions one can pursue to further the study of theories. Let's mention a couple of them. Firstly it is worthwhile to carry out a comparative research into theories. Honda and Yoshida [HY95] have introduced a different notion of theory. They extended the notion of λ-theory [Bar84] to the framework of process models. A theory in their sense is a set of equalities closed under the process operations of the frame calculus. Their starting point is more algebraic than logical, whereas our motivation is less algebraic than logical. It is apparent that a theory in the present setting is a theory of Honda and Yoshida. Unlike in their treatment, the algebraic property of our theories comes for free. It would be interesting to investigate the relationship between these two approaches.

Secondly it is interesting to look for complete proof systems for general theories. Suppose $\mathbf{A}$ is a theory. Let $AS_{\mathbf{A}}$ be obtained by combining AS with the following laws for $\mathbf{A}$.

$$A = \mathbf{0}, \text{ for every } A \in \mathbf{A}. \tag{6}$$

For which recursive theories for instance is $AS_{\mathbf{A}}$ complete? It is easy to come up with theories for which $AS_{\mathbf{A}}$ does not appear to be complete. Take for instance the theory $\mathbf{A_a} = \{\overline{a}(c).\overline{a}(c)\}$. It is unlikely that $AS_{\mathbf{A}_a} \vdash \overline{a}(c) = \mathbf{0}$. In fact it follows from Sewell's nonaxiomatisability result [Sew97] that it must be incomplete if the frame calculus is CCS. It remains to check if this negative result is also valid when the frame calculus is π. A more difficult task is to answer the question if there exists a recursive theory $\mathbf{A}$ such that $\simeq_{\mathbf{A}}$, when restricted to the finite terms, has no complete recursive equational systems whatsoever. These are issues to be investigated in future.

Acknowledgments

This work has been supported by NSFC (60873034). The author is grateful to the anonymous referees for their comments and suggestions.

References

[Abr93] Abramsky, S.: Computational interpretations of linear logic. Theoretical Computer Science 111, 3–57 (1993)

[ACS98] Amadio, R., Castellani, I., Sangiorgi, D.: On bisimulations for the asynchronous π-calculus. Theoretical Computer Science 195, 291–324 (1998)

[Bar84] Barendregt, H.: The lambda calculus: Its syntax and semantics. Studies in Logic and Foundations of Mathematics (1984)

[Bou92] Boudol, G.: Asynchrony and the π-calculus. Technical Report RR-1702, INRIA Sophia-Antipolis (1992)

[CF10] Cai, X., Fu, Y.: The λ-calculus in the π-calculus (2010)

[Fu10] Fu, Y., Lu, H.: On the expressiveness of interaction. Theoretical Computer Science 411, 1387–1451 (2010)

[FY03] Fu, Y.: Theory of interaction (2010)

[FY03] Fu, Y., Yang, Z.: Tau laws for pi calculus. Theoretical Computer Science 308, 55–130 (2003)

[FZ10] Fu, Y., Zhu, H.: The name-passing calculus (2010)

[HO95] Hyland, M., Ong, L.: Pi-calculus, dialogue games and pcf. In: Proc. 7th ACM Conference on Functional Programming Languages and Computer Architecture (FPCA 1995), pp. 96–107 (1995)

[HT91a] Honda, K., Tokoro, M.: An object calculus for asynchronous communications. In: America, P. (ed.) ECOOP 1991. LNCS, vol. 512, pp. 133–147. Springer, Heidelberg (1991)

[HT91b] Honda, K., Tokoro, M.: On asynchronous communication semantics. In: Tokoro, M., Wegner, P., Nierstrasz, O. (eds.) ECOOP-WS 1991. LNCS, vol. 612, pp. 21–51. Springer, Heidelberg (1992)

[HY95] Honda, K., Yoshida, M.: On reduction-based process semantics. Theoretical Computer Science 151, 437–486 (1995)

[Lin95] Lin, H.: Complete inference systems for weak bisimulation equivalences in the π-calculus. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 187–201. Springer, Heidelberg (1995)

[Mil89] Milner, R.: Communication and Concurrency. Prentice Hall, Englewood Cliffs (1989)

[Mil92] Milner, R.: Functions as processes. Mathematical Structures in Computer Science 2, 119–146 (1992)

[MPW92] Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes. Information and Computation 100, 1–40 (Part I), 41–77 (Part II) (1992)

[PS95] Parrow, J., Sangiorgi, D.: Algebraic theories for name-passing calculi. Information and Computation 120, 174–197 (1995)

[Sew97] Sewell, P.: Nonaxiomatisability of equivalence over finite state processes. Annals of Pure and Applied Logic 90, 163–191 (1997)

[Wal95] Walker, D.: Objects in the π-calculus. Information and Computation 116, 253–271 (1995)

On the Compositionality of Round Abstraction

Dan R. Ghica* and Mohamed N. Menaa

University of Birmingham, U.K.

Abstract. We revisit a technique called *round abstraction* as a solution to the problem of building low-latency synchronous systems from asynchronous specifications. We use a trace-semantic setting akin to Abramsky's *Interaction Categories*, which is also a generalisation of pointer-free game semantic models. We define partial and total correctness for round abstraction relative to composition and note that in its most general case, round abstraction can lead to incorrect behaviour. We then identify sufficient properties to guarantee partially correct composition. Finally, we propose a framework for round abstraction that is totally correct when applied to asynchronous behaviours. We apply this procedure to the *Geometry of Synthesis*, a technique for compiling higher-order imperative programming languages into digital circuits using game semantics.

1 Introduction

Concurrency models can be qualified as either synchronous or asynchronous. Synchrony is typically characterised by such notions as *simultaneous occurrence* and *instantaneous communication* – concepts that cannot be found in asynchronous models. Application areas for the two models have typically been different: asynchronous concurrency is used when bounds on the time necessary for interaction cannot be guaranteed (e.g. distributed systems), or when time is intentionally abstracted (e.g. concurrent high-level programming languages), whereas synchronous concurrency is commonly used when time is an essential facet of the system (e.g. safety-critical systems or digital circuits).

Studying the correlation of synchrony and asynchrony has been an object of research for a long time. Milner was the first to establish that asynchronous computation can be modelled using a synchronous calculus (SCCS) [1], also showing how the asynchronous Calculus of Communicating Systems (CCS) [2] can be derived from SCCS. Later work showed similar results in varied contexts [3,4]. However, the naive representation of an asynchronous process as a synchronous one is inefficient and deriving a *low-latency* synchronous system from an asynchronous one is arguably more difficult. Even recovering synchronicity after it is removed from a specification is a non-trivial procedure [4].

In a seminal paper, Alur and Henzinger describe a more general approach based on a specification language called *Reactive Modules* [5]. This technique, called *round abstraction*, allows arbitrarily many computational steps to be

* Supported by EPSRC Advanced Research Fellowship EP/D070880/1.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 417–431, 2010.

viewed as a single macro-step. Intuitively, this is achieved by using a designated set of events as a clock, and considering any events that occur between two ticks as being simultaneous. This forms the basis for an elegant solution to the problem of building synchronous systems from asynchronous specifications. Round abstraction is essentially an approximation technique which removes some of the timing information between events in a process. The original formulation of round abstraction is monolithic and applies to whole systems, not addressing the question of whether round-abstracted systems still interact correctly with each other.

Contributions. This paper is a study of the compositional properties of round abstraction. We present trace-based models of synchronous and asynchronous concurrency, verifying that they have a reasonable mathematical structure, and describe a generalised notion of round abstraction in this setting. We formally define two notions of correctness, partial and total, and examine their behaviour relative to composition. For total correctness we propose a characterisation of round abstractions that compose correctly.

Round abstraction is an important technique and the study of its mathematical properties is interesting in general. However, we are interested in particular in applying this technique within the framework of *Geometry of Synthesis*, the automatic synthesis of digital circuits from programming languages via their game-semantic models [6,7,8]. Game-semantic models of concurrent languages are typically asynchronous [9,10] whereas the typical implementation of digital circuits is clocked synchronous. Round abstraction, in this context, offers a way to generate provably correct synchronous circuits from asynchronous pointer-free game models. The current work raises further questions about generalising the properties, models and criteria we rely on, but we use compatibility with the game semantic setting and applicability to hardware synthesis as a pragmatic assessment to conclude that our results are reasonable and useful. To further this point, we illustrate this application with the non-trivial example of synthesising a circuit for iteration with very low latency, which can also handle instant feedback on all its inputs.

2 A Trace Model of Processes

We first introduce a trace model of concurrency which is a slight generalisation of the game models for concurrency [11,10]. The metaphor is one of black boxes that interact using simple connectors ("wires"). A box has a port structure ("signature") and its behaviour is a set of traces of input and output events.

Definition 1 (Signature). *A signature A is a finite set together with a labelling function and a causality relation. Formally, it is a triple $\langle L_A, \pi_A, \vdash_A\rangle$ where*

- L_A *is a* set of port labels*,*
- $\pi_A : L_A \to \{i, o\}$ *maps each label to an* input/output polarity*,*
- $\vdash_A$ *is the transitive reduction of a partial order on* L_A *called* causality*, such that if $a \vdash b$ then $\pi_A(a) \neq \pi_A(b)$.*

Signatures are akin to game-semantic *arenas*. The causality relation, akin to game-semantic *enabling*, will turn out to be technically important. Intuitively, it models the fact that a b-event cannot happen unless some a-event causes it. Note that causality is both descriptive, when an input causes an output, but also prescriptive, when an output can require the environment to only produce certain inputs. We denote by π^* a labelling function which is like π but has the input/output polarities reversed; similarly for A^*. We denote by I_A the minimal elements of $\vdash_A$.

Definition 2 (Locally synchronous trace). *A* trace *over a signature A is a triple $\langle E, \preceq, \lambda\rangle$ where E is a finite set of events, $\preceq$ is a total preorder on E and $\lambda : E \to A$ is a function mapping events to labels in A.*

The total preorder signifies temporal precedence; for an element $e \in E$, if $\lambda(e) = a \in L_A$ we say that e is an *occurrence* of a. Traces are equivalent if there is a bijection between their carrier sets acting homomorphically on event labelling and temporal ordering; in practice, we work with traces as equivalence classes, as the choice of the carrier set E is irrelevant. It is convenient to define

Definition 3 (Simultaneity). *Given $t = \langle E, \preceq, \lambda\rangle$ over A, we say that two events are* simultaneous*, written $e_1 \approx e_2$ if and only if $e_1 \preceq e_2$ and $e_2 \preceq e_1$.*

Our conception of synchrony is a minimalistic one; time is discretised and events can be simultaneous, which is the essential feature of a synchronous process [12]. However, our notion of trace does not rely on a global clock. Instead, we rather assume that each system has its own internal and abstract clock, relative to which simultaneity is defined, and that these clocks can compose. The notion of synchrony we have is a local one [13].

Example 1. For illustration, we will often informally use a simplified notation for these traces whereby a trace such as $\langle\{e_1, e_2, e_3, e_4\}, \{(e_1, e_2), (e_1, e_3), (e_1, e_4), (e_2, e_3), (e_2, e_4), (e_3, e_4), (e_4, e_3)\}, \lambda\rangle$ with $\lambda = \{e_1 \mapsto a, e_2 \mapsto b, e_3 \mapsto a, e_4 \mapsto c\}$ is simply denoted by $a.b.\langle ac\rangle$. The trace consists of an a-event, followed by a b-event, followed by an a-event and a c-event at the same time.

We focus on a particular kind of traces, which satisfy the following principle:

Definition 4 (Singularity). *The events of a trace $\langle E, \preceq, \lambda\rangle$ over signature A are* singular *if and only if for any two events $e_1, e_2 \in E$, if $e_1 \approx e_2$ and $\lambda(e_1) = \lambda(e_2)$ then $e_1 = e_2$.*

A trace has singular events if it does not have any distinct simultaneous occurrences of the same event. This restriction is not inherent to synchronous concurrency but is essential for modelling low level circuit behaviour where events are not implicitly buffered or tagged. By definition, we rule out phenomena akin to *schizophrenia* in Esterel [12]. We denote by $\Theta(A, B, \ldots)$ the set of such traces over $A + B + \ldots$

On the other hand, an asynchronous trace is a trace where the simultaneity relations between two events is equal to the identity:

Definition 5 (Asynchronous trace). *A trace $\langle E, \preceq, \lambda \rangle$ over signature A is* asynchronous *if and only if $\preceq$ is a total order.*

The definition above reflects the failure of synchrony in asynchronous systems, as no two distinct events can be ascertained to occur precisely at the same time.

Another, more technical condition, which is inspired by game semantics and also reflects the low-level nature of the systems we model is *serial causation.*

Definition 6 (Serial causation). *In a trace $\langle E, \preceq, \lambda \rangle$ over signature A, we say that an event $e_1 \in E$ is the* actual cause *of $e_0 \in E$, written $e_1 \curvearrowleft e_0$, if and only if $\lambda(e_1) \vdash \lambda(e_0)$ and for any e_2 such that $e_1 \preceq e_2 \preceq e_0$, $\lambda(e_2) \nvdash \lambda(e_0)$.*

If two ports are causally related in a signature, then serial causation assigns the most recent event which can cause another event as its actual cause. Actual causation, which is an event-level relation, must be determined in order to define asynchronous behaviour properly. This is because the order of the occurrence of events must be closed under certain permutations that are not allowed to swap an event and its cause. In higher-level systems, such as games or data flow [14], causality can be encoded directly, as justification pointers or token tags, respectively. In a lower-level system, it is necessary to be able to recover this information implicitly from the structure of the trace. Note that, in certain game models, justification pointers can also be recovered from the structure of the play [15].

We define the *concatenation* of two traces at the level of rounds, i.e., all events of the second trace come after the events of the first trace.

Definition 7 (Trace concatenation). *The* concatenation *of two traces $s = \langle E, \preceq_s, \lambda_s \rangle$, $t = \langle F, \preceq_t, \lambda_t \rangle$, denoted by $s \cdot t$, is the trace defined by the triple $\langle E + F, \preceq_s + \preceq_t + (E \times F), \lambda_s + \lambda_t \rangle$.*

Definition 8 (Process). *A* process *σ over signature A, $\sigma : A$, is a prefix-closed set of traces, i.e. $\forall s, t \in \Theta(A)$ if $s \cdot t \in \sigma$ then $s \in \sigma$.*

For an arbitrary set of traces τ, let $pc(\tau)$ be the smallest process including τ.

Traces of an asynchronous process must be closed under certain permutations of events, corresponding to inputs occurring earlier and outputs occurring later. To maintain consistency, we require that the serial causation relation between events is not changed by the permutations. This is a reformulation of a saturation principle which is common in game models for asynchronous languages [16,10]. Let $\lesssim$ on $\Theta(A)$ be defined as the least reflexive transitive relation such that $s' \lesssim s$ if and only if

1. (a) If e is an input then $s' = s_0 \cdot e \cdot s_1 \cdot s_2$ and $s = s_0 \cdot s_1 \cdot e \cdot s_2$, or
   (b) If e is an output then $s' = s_0 \cdot s_1 \cdot e \cdot s_2$ and $s = s_0 \cdot e \cdot s_1 \cdot s_2$
2. There exists a bijection $\phi : E_s \simeq E_{s'}$ so that for any events such that $e_1 \curvearrowleft_s e_2$ we have $\phi(e_1) \curvearrowleft_{s'} \phi(e_2)$ and $\lambda_s(e_i) = (\lambda_{s'} \circ \phi)(e_i)$ for $i \in \{1, 2\}$.

Definition 9 (Asynchronous process). *An* asynchronous process *over signature A, written $\sigma : A$, is a prefix and $\lesssim$-closed set of asynchronous traces, i.e. if $s \in \sigma$ and $s' \lesssim s$ then $s' \in \sigma$.*

Let $s \upharpoonright A$ be the trace obtained from s by deleting all events with labels not belonging to L_A. Composition of processes is defined similarly to game semantic composition, by synchronisation followed by hiding.

We define a composite arena in a way that is similar to the game-semantic exponential: $A \to B$ is $\langle L_A + L_B, \pi_A^* + \pi_B, \vdash_A + \vdash_B + I_B \times I_A \rangle$.

Definition 10 (Composition). *For $\sigma : A \to B$ and $\tau : B \to C$ interaction is defined as $\sigma \between \tau = \{t \in \Theta(A, B, C) \mid t \upharpoonright A + B \in \sigma \wedge t \upharpoonright B + C \in \tau\}$, and composition as $\sigma ; \tau : A \to C = \left\{t \upharpoonright A + C \mid t \in \sigma \between \tau \right\}$.*

The results below indicate the formalism so far makes sense, situating us in a framework similar to Abramsky's Interaction Categories [17].

Theorem 1. *1. Processes form a Compact Closed Symmetric Monoidal Category, which we call $\mathcal{ST}$.*
2. Asynchronous processes form a Compact Closed Symmetric Monoidal Category, which we call $\mathcal{AT}$.

Note that although asynchronous processes are a subset of the more general notion of (synchronous) processes, they do not form a sub-category. The identity for processes in general is synchronous, instantly replicating any input at one end as an output on the other. Physically, it corresponds to a set of wires directly connecting input and output. Conceptually, it is an instantaneous version of the game semantic copycat strategy. Therefore, in general, identity cannot be an asynchronous process. However, asynchronous processes have their own notion of identity, similar to the copycat strategy in asynchronous games.

2.1 From Local to Global Synchrony

This section is somewhat of an aside to the main thrust of the paper. In it, we show that the local synchrony assumption is expressive enough to construct globally synchronous systems in a canonical way, using a *clock monad*. Therefore, the results in this paper can be extended in a straightforward way to systems using external and explicit clocks, which are the predominant digital design paradigm.

Let functor $T : \mathcal{ST} \to \mathcal{ST}$ be defined as $T(A) = A \otimes Ck$, $T(f) = f \otimes id_{Ck}$, where Ck is a reserved one-port object. Let natural transformation (at object A) $\eta_A : A \to T(A) = A \to A \otimes Ck$ be defined as

$$\eta_A = \{s \in \Theta(A, A', Ck) \mid s \upharpoonright (A, A') \in id_A\}$$

Let natural transformation (at object A) $\mu_A : T^2(A) \to T(A) = A \otimes Ck \otimes Ck \to A \otimes Ck$ be defined as

$$\begin{aligned}\mu_A = \{&s \in \Theta(A, Ck, Ck', A', Ck'') \mid \\ &s \upharpoonright (A, A') \in id_A \text{ and } s \upharpoonright (Ck, Ck') \in id_{Ck} \text{ and } s \upharpoonright (Ck, Ck'') \in id_{Ck}\}\end{aligned}$$

These behaviours correspond to the circuit constructions in Fig. 1. Signature A has an arbitrary number of ports, but Ck is single-port; we show its input/output polarity with arrows for extra clarity.

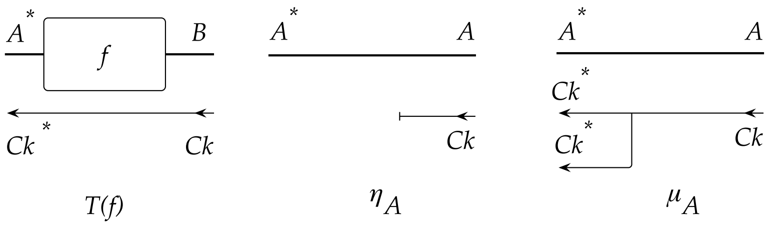
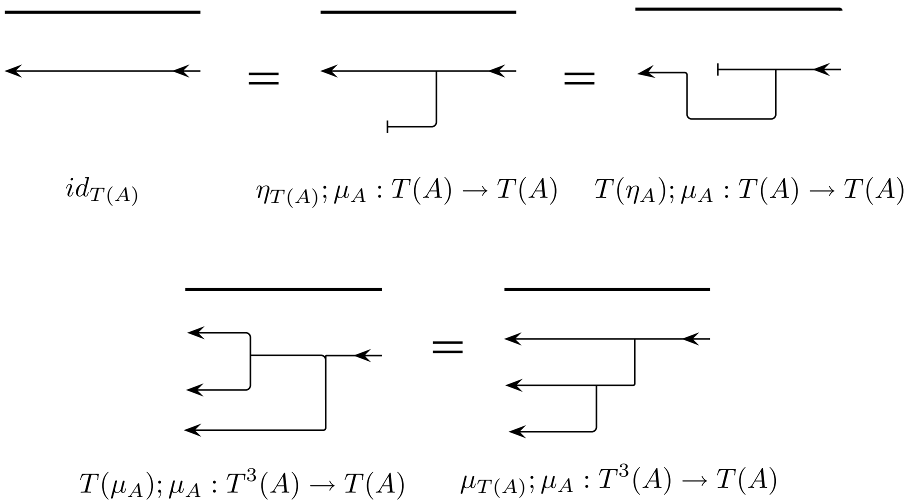


Fig. 1. Clock monad


$id_{T(A)}$ $\eta_{T(A)}; \mu_A : T(A) \to T(A)$ $T(\eta_A); \mu_A : T(A) \to T(A)$

$T(\mu_A); \mu_A : T^3(A) \to T(A)$ $\mu_{T(A)}; \mu_A : T^3(A) \to T(A)$

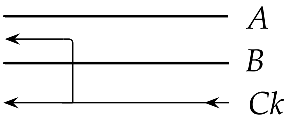
Fig. 2. Circuit representation of the clock monad axioms

Proposition 1. *$\langle T, \eta, \mu \rangle$ is a monad.*

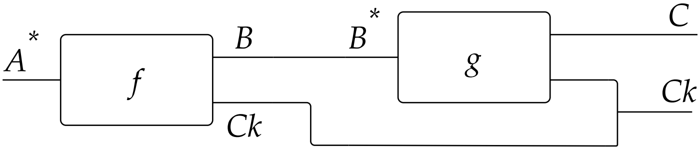
Proof. The coherence axioms for the monad correspond to showing that the circuits in Fig. 2 are behaviourally equal. Because we are in a symmetric compact closed category, in general graph-isomorphic circuits will be behaviourally equal.

Proposition 2. *The clock monad is a commutative strong monad.*

Proof. The strength of the monad exists, trivially, as $TA \otimes B \simeq T(A \otimes B)$. Both sides of the commutativity equation $t_{TA,B}; T(t'_{A,B}); \mu_{A\otimes B} = t'_{A,TB}; T(t_{A,B}); \mu_{A\otimes B}$ correspond to constructing the circuit below



The existence of the strong monad corresponds to driving two circuits from the same clock source in a unique way. The strong monad shows how clocked processes can be composed in parallel. Using the clock monad we can use the associated Kleisli category to compose clocked circuits sequentially. Using the standard definitions, the Kleisli composition of clocked circuits $f : A \to TB$, $g : B \to TC$ corresponds to the diagram below



$$f;_T g = f; Tg; \mu_C$$

Note that the construction is flexible and general enough to allow the definition of different clock domains using different clocks and clock monads. Using the clock monad, we can identify a sub-category of $\mathcal{ST}$ which consists of *deterministic* processes. The details of this are outside the scope of this paper.

3 Round Abstraction

Round abstraction was introduced by Alur and Henzinger as part of their specification language *Reactive Modules* [5]. It is a technique that introduces a multiform notion of computational step, allowing arbitrarily many events to be viewed as a discrete *round*. In a reactive module, rounds are delineated using a set of observable events as a clock, in such a way that any computational steps occurring between two clock actions are observed as a single one. Its importance lies in that it defines a simple technique for constructing synchronous systems from asynchronous ones. Indeed, it does this by introducing a clock, such that the notion of a round is based solely on the input/output behaviour of a system. We will avoid the choice of a clock, which is arbitrary, and thus provide a more general notion of round abstraction.

At this point, it may be observed that locally-synchronous traces have an inherent *order of synchronicity*. For example, for each possible succession of unique events, the least synchronous trace is the one where they occur one after the other, where the most synchronous trace is the one where they all occur simultaneously.

Definition 11 (Round abstraction on traces). *Let $s = \langle E_s, \preceq_s, \lambda_s \rangle$, $t = \langle E_t, \preceq_t, \lambda_t \rangle$ be traces. We say that t* is a round abstraction of *s, written $s \sqsubseteq t$, if and only if $\langle E_s, \lambda_s \rangle$ and $\langle E_t, \lambda_t \rangle$ are ϕ-isomorphic and ϕ is monotonic relative to temporal ordering, i.e. for any $e, e' \in E_s$, if $e \preceq_s e'$ then $\phi(e) \preceq_t \phi(e')$.*

Note that this immediately implies simultaneity is preserved, i.e. if $e \approx_s e'$ then $\phi(e) \approx_t \phi(e')$. The converse is obviously false since round abstraction can make non-simultaneous events in s simultaneous in t.

In order to appreciate the challenges we need to address, let us first informally introduce two desirable correctness criteria on behaviours, which we shall call "partial" and "total" round abstraction. Partial round abstraction $\sigma \sqsubseteq \tau$ requires that τ, the "abstracted" behaviour, has no "junk" traces which do not originate from σ. A stronger property, total round abstraction $\sigma \underset{\approx}{\sqsubset} \tau$, additionally has that all behaviour of σ can be found, in an abstracted form, in τ. Either notion of round abstraction of processes is, as is usually the case with abstraction, trivially compositional in the sense that if $\sigma \sqsubseteq \tau, \tau \sqsubseteq \gamma$ then $\sigma \sqsubseteq \gamma$ and if $\sigma \underset{\approx}{\sqsubset} \tau, \tau \underset{\approx}{\sqsubset} \gamma$ then $\sigma \underset{\approx}{\sqsubset} \gamma$. But it is not the case that $\sigma \sqsubseteq \sigma', \tau \sqsubseteq \tau'$ implies $\sigma;\tau \sqsubseteq \sigma';\tau'$, or $\sigma \underset{\approx}{\sqsubset} \sigma', \tau \underset{\approx}{\sqsubset} \tau'$ implies $\sigma;\tau \underset{\approx}{\sqsubset} \sigma';\tau'$. This is similar to the non-compositionality of abstract interpretation [18]. As immediate counter-examples, consider the following.

Example 2. Let $\sigma, \sigma' : A \to B$ and $\tau, \tau' : B \to C$, with $L_A = \{a\}$, $L_B = \{b_1, b_2\}$, $L_C = \{c\}$, be the following processes:

$$\begin{array}{lcl} \sigma = pc(\{a.b_1.b_2\}) & \sqsubseteq & \sigma' = pc(\{a.\langle b_1 b_2\rangle\}) \\ \tau = pc(\{b_2.b_1.c\}) & \sqsubseteq & \tau' = pc(\{\langle b_2 b_1\rangle.c\}) \end{array}$$

We then have $\sigma;\tau = pc(\{a\})$ but $\sigma';\tau' = pc(\{a.c\}) \not\sqsupseteq \sigma;\tau$.

Example 3. Let $\sigma, \sigma' : A \to B$ and $\tau, \tau' : B \to C$, with $L_A = \{a\}$, $L_B = \{b_1, b_2\}$, $L_C = \{c\}$, be the following processes:

$$\begin{array}{lcl} \sigma = pc(\{a.b_1.b_2\}) & \underset{\approx}{\sqsubset} & \sigma' = pc(\{\langle a b_1 b_2\rangle\}) \\ \tau = pc(\{b_1.c.b_2.c\}) & \underset{\approx}{\sqsubset} & \tau' = pc(\{\langle b_1 c\rangle.\langle b_2 c\rangle\}) \end{array}$$

Then we have $\sigma;\tau = pc(\{a.c.c\})$ but $\sigma';\tau' = \{\epsilon\} \not\underset{\approx}{\sqsupset} \sigma;\tau$.

In these examples, and typically, the way *deadlock* is handled will play the key role, because round abstraction can both resolve and introduce deadlocks.

In Ex. 2, the two behaviours do not compose well because the order in which b_1, b_2 are issued by σ does not coincide with the order in which they can be received by τ; round abstraction makes the two events simultaneous and thus solves the deadlock. In Ex. 3 round abstraction requires the two B events to be simultaneous in σ' and consecutive in τ' therefore introducing deadlock.

4 Partial Correctness

Given a trace v, let $\varPi(v)$ be the set of its *non-identity permutations*, i.e. the set of traces with the same events but a different temporal order. In order to prevent round abstraction resolving deadlocks, as in Ex. 2, we introduce the following condition.

Definition 12 (Compatibility). *Two processes $\sigma_1 : A_1 \to B, \sigma_2 : B \to A_2$, are said to be* compatible*, written $\sigma_1 \asymp \sigma_2$, if and only if for all $v \in \varTheta(A_1, B, A_2)$ if $v \upharpoonright A_i, B \in \sigma_i$ and there is a permutation $p \in \varPi(v)$ such that $p \upharpoonright B, A_j \in \sigma_j$ then $v \upharpoonright B, A_j \in \sigma_j$, for $i, j \in \{1, 2\}$, $i \neq j$.*

This requirement ends up ensuring partial correctness almost by definition. Its merit is rather as a characterisation of the main cause of failure of composition for partial round abstraction. Going back to Ex. 2, the problem trace is $v = a.b_1.b_2.c$ and the problem permutation is $p = a.b_2.b_1.c$.

Definition 13 (Partial round abstraction). *Let σ, τ be processes over A. We say that τ is a* partial round abstraction *of σ, written $\sigma \sqsubseteq \tau$, if and only if for any $t \in \tau$ there is $s \in \sigma$ such that $s \sqsubseteq t$.*

In a partial round abstraction, the abstracted process does not contain any "junk" traces which do not correspond to traces in the original process. However, it is possible for some traces in the original process to have no corresponding trace in its abstraction.

One of our main results is the soundness of composition relative to partial round abstraction, formulated as follows:

Theorem 2 (Soundness). *For any two* compatible asynchronous *processes $\sigma : A \to B$ and $\tau : B \to C$, with round abstractions σ', τ' respectively, if $\sigma \sqsubseteq \sigma'$ and $\tau \sqsubseteq \tau'$ then $\sigma; \tau \sqsubseteq \sigma'; \tau'$.*

Note that the asynchrony requirement is not necessary and soundness can be immediately generalised to processes in general. The key property we use in this theorem is compatibility, which is in fact sufficient to guarantee soundness.

5 Total Correctness

For ensuring total correctness, another notion of "good" composition of processes is required. Consider the following example.

Example 4. Let $\sigma, \sigma' : A \to B$ and $\tau, \tau' : B \to C$, with $L_A = \{a\}$, $L_B = \{b_1, b_2, b_3\}$, $L_C = \{c\}$, be $\sigma = pc(\{a.b_1.b_2\}) \underset{\sim}{\sqsubset} \sigma' = pc(\{\langle ab_1b_2 \rangle\})$ and $\tau = pc(\{b_1.b_3.c\}) \underset{\sim}{\sqsubset} \tau' = pc(\{\langle b_1b_3c \rangle\})$. Then we have $\sigma; \tau = pc(\{a\})$ but $\sigma'; \tau' = \{\epsilon\} \not\underset{\sim}{\sqsupset} \sigma; \tau$.

In this example, the original processes σ and τ compose well up to b_1 then deadlock as they attempt to synchronise on mismatched events. Because the abstracted processes σ', τ' are single-round processes the failure of composition prevents the creation of any complete rounds, therefore it produces only the empty-trace process as a result. To avoid this situation we only consider processes that compose *safely*, i.e. can handle each other's events.

Definition 14. *Given a trace $u \in \sigma : A$ we define its* next-action *set $next_\sigma(u) = \{a \in L_A \mid u \cdot e \in \sigma, \lambda(e) = a\}$.*

We define $\text{next}^i_\sigma(u)$, $\text{next}^A_\sigma(u)$ or $\text{next}^{A,i}_\sigma(u)$ as the obvious restrictions of the next-action set to inputs (or outputs) or a sub-set of ports or both. A safe interaction is one in which the outputs of one of the circuits can be handled by the other as input and vice versa.

Definition 15 (Safety). *Two asynchronous processes $\sigma_1 : A_1 \to B$, $\sigma_2 : B \to A_2$, are said to compose* safely, *written $\sigma_1 \smile \sigma_2$, if and only if for any interaction trace $u \in \sigma_1 \lightning \sigma_2$, $\mathrm{next}^{o,B}_{\sigma_i}(u \restriction A_i, B) \subseteq \mathrm{next}^{i,B}_{\sigma_j}(u \restriction B, A_j)$ for $i, j \in \{1, 2\}, i \neq j$.*

Our definition states that the composition of two processes is "unsafe" if one of them produces output that cannot be handled by the other. This is a generalisation of the notion of Opponent-completeness in game semantics, the requirement that a strategy must handle any (legal) move of the Opponent, and it is also an example of Vardi's "principle of comprehensive modelling" [19]. Finally, this formulation of safety is also justifiable in a low-level view of circuits, where events cannot be buffered but must be processed as they occur.

Total round abstraction requires not only that the abstracted process is junk-free, but also that no behaviour is lost.

Definition 16 (Total round abstraction). *Let $\sigma : A \to B$ be an asynchronous process and $\sigma' : A \to B$ be a process. We say that σ' is a* total round abstraction *of σ, written $\sigma \underset{\sim}{\sqsubset} \sigma'$ if and only if $\sigma \sqsubseteq \sigma'$ and for any $s \in \sigma$ there exist $s_0 \in \sigma$, $w \in \varTheta(A, B)$ and $s' \in \sigma'$ such that $s_0 \lesssim s$ and $s_0 \cdot w \sqsubseteq s'$.*

Total round abstraction has a more complicated technical definition because prefix-closure is defined at the level of rounds rather than at the level of events. It says that for any original trace, another trace can be found within its saturation closure and then "padded" with some events so that it matches an abstracted trace. The reason is that, for an asynchronous trace, prefix-closure will generate more prefixes than in its synchronous, round-abstracted trace; however, we want round abstraction to automatically extend to prefixes. For example, at the level of traces, $a.b.c \sqsubseteq \langle abc \rangle$ but $pc(a.b.c) = \{\epsilon, a, a.b, a.b.c\}$ whereas $pc(\langle abc \rangle) = \{\epsilon, \langle abc \rangle\}$; using the definition above $pc(a.b.c) \underset{\sim}{\sqsubset} pc(\langle abc \rangle)$.

In general, total round abstraction is not preserved even in the case of compatible safely-compositional asynchronous processes because events may be assigned to rounds in a way that prevents proper composition. This is the typical situation presented in Ex. 3. We introduce additional criteria for total round abstraction to guarantee correctness under composition.

Let us first define the ancillary concept of *trace fusion*, which is like concatenation but the final round of the first trace and the initial round of the second trace are taken to be simultaneous. Let the last round of a trace $s = \langle E, \preceq_s, \lambda_s \rangle$ be defined in the obvious way, $last(s) = \{e \in E \mid \forall e' \in E.e' \preceq e\}$. The *first* round is defined in an analogous way.

Definition 17 (Trace fusion). *The* fusion *of two traces $s = \langle E, \preceq_s, \lambda_s \rangle$, $t = \langle F, \preceq_t, \lambda_t \rangle$, denoted by $s * t$, is the trace defined by the triple $\langle E + F, \preceq', \lambda_s + \lambda_t \rangle$, where $\preceq' = \preceq_s + \preceq_t + E \times F + first(t) \times last(s)$.*

The concept below is one of the key contributions of this paper, establishing a framework in which total round abstraction composes well:

Definition 18 (Receptive round abstraction). *Let $\sigma : A$ be an asynchronous process. Process σ' is a* receptive round abstraction *of σ, written $\sigma \underset{\approx}{\sqsubset} \sigma'$, if and only if $\sigma \underset{\sim}{\sqsubset} \sigma'$ and for any distinct inputs i, i_1, i_2 and output o*

1. *if $s_0 \cdot i_1 \cdot i_2 \in \sigma$ then there exists traces of shape $s_0' \bullet i_1 \cdot i_2 \bullet s_1$ and $s_0' \bullet i_1 * i_2 \bullet s_1$ in σ',*
2. *if $s_0 \cdot o \cdot i \in \sigma$ then there exists traces of shape $s_0' \bullet o \cdot i \bullet s_1$ and $s_0' \bullet o * i \bullet s_1$ in σ'.*

Moreover,

1. *if $t_0 \cdot r_0 * i_1 \cdot i_2 * r_1 \cdot t_1 \in \sigma'$ and $t' = t_0 \cdot r_0 * i_1 * i_2 * r_1 \cdot t_1$ is well formed then $t' \in \sigma'$,*
2. *if $t_0 \cdot r_0 * o \cdot i * r_1 \cdot t_1 \in \sigma'$ and $t' = t_0 \cdot r_0 * o * i * r_1 \cdot t_1$ is well formed then $t' \in \sigma'$,*

where each instance of $\bullet$ stands for concatenation $\cdot$ or fusion $*$ and $s_0 \sqsubseteq s_0'$.

Note that well-formedness implies that singularity and serial causation are respected. The conditions above are the formalisation of the following requirements:

Input receptivity: successive inputs can be received in succession as well as simultaneously,

Instant feedback receptivity: an input following an output may also be received simultaneously.

In essence, these rules stipulate that the environment can produce input either instantly or later, and the system must handle both situations. Now we can introduce our main result, stating that receptive round abstraction is compositional: it preserves both total correctness and receptivity.

Theorem 3 (Adequacy). *For any two compatible safely-compositional asynchronous processes $\sigma : A \to B$ and $\tau : B \to C$, composition preserves receptive round abstraction: if $\sigma \underset{\approx}{\sqsubset} \sigma'$ and $\tau \underset{\approx}{\sqsubset} \tau'$ and $\sigma \asymp \tau$ and $\sigma \smile \tau$ then $\sigma;\tau \underset{\approx}{\sqsubset} \sigma';\tau'$.*

We illustrate the theorem with a simple example showing the essential use of allowable permutations for asynchronous traces, both in Def. 16 and in the proof of this theorem. Let $\sigma : A \to B$, $\tau : B \to C$ be asynchronous processes and $\sigma' : A \to B$, $\tau' : B \to C$ be processes, with $L_A = \{a\}$, $L_B = \{b_1, b_2\}$, $L_C = \{c\}$ and $\sigma = pc(\{b_1^i.a.a.b_2^i\}) \underset{\sim}{\sqsubset} \sigma' = pc(\{\langle b_1^i a\rangle.\langle a.b_2^i\rangle\})$, $\tau = pc(\{b_1^o.b_2^o.c\}) \underset{\sim}{\sqsubset} \tau' = pc(\{\langle b_1^o b_2^o c\rangle\})$. The traces $\langle b_1^i a\rangle\langle a b_2^i\rangle$ and $\langle b_1^o b_2^o c\rangle$ do not compose because events b_1, b_2 must be placed in different rounds in the first trace (due to the singularity of a) and are in the same round in the second. However, since σ is an asynchronous process, we know that it must also contain the trace $b_1^i.b_2^i.a.a$ generated by saturation (by the definition of $A \to B$, A-events cannot cause B-events). This trace is round abstracted to (for example) $\langle b_1^i b_2^i\rangle.a.a$ which composes well with $\langle b_1^o b_2^o c\rangle$.

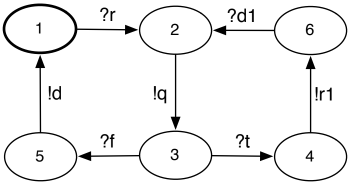


Fig. 3. Asynchronous model of game-semantic iteration

6 Application to the Geometry of Synthesis

GoS [6] is a semantics-directed approach to solving the long-standing problem of synthesising hardware from higher-level behavioural descriptions written in programming languages, i.e. “hardware compilation.” The basic idea of GoS is to consider the game-semantic model of a programming language as an input/output behavioural specification for a digital circuit. If the circuit implementation of this specification is also asynchronous then the game model can be mapped almost directly, and efficiently, into asynchronous designs [20].

The question of mapping game semantics into synchronous designs is, for reasons elaborated in this paper, more complex since game models are asynchronous. A straightforward mapping of the asynchronous model into synchronous circuits is possible but naive; it is expensive in terms of circuit footprint and it has unacceptable high latency.

The game model for constants such as skip (the empty command), 0 and 1 consists of asynchronous processes $\llbracket \mathsf{skip} \rrbracket = pc(\{q.a\}), \llbracket 0 \rrbracket = pc(\{q.t\}), \llbracket 1 \rrbracket = pc(\{q.f\})$. A full definition of the game semantic model is out of the scope of this paper, but we can give some basic intuitions. A constant is an initial “request” input followed by an acknowledging output indicating the value. For skip this is just a token indicating that the command completed successfully.

6.1 Implementation

The naive mapping of these processes into circuits requires two-state automata to read the input q *then* on the next cycle to produce the output and reset to the initial state. This means that any use of a constant will require two flip-flops, some combinatorial logic and will introduce a unit delay. On the other hand, the most efficient receptive round abstraction will give the following *synchronous* representations: $\llbracket \mathsf{skip} \rrbracket \underset{\approx}{\sqsubset} pc(\{\langle qa \rangle\}), \llbracket 0 \rrbracket \underset{\approx}{\sqsubset} pc(\{\langle qt \rangle\}), \llbracket 1 \rrbracket \underset{\approx}{\sqsubset} pc(\{\langle qf \rangle\})$.

These representations are now stateless, instantly propagating the input q to the output. As circuits, they can be implemented simply as connectors. This is in itself a very useful optimisation, but it requires the round abstraction of all the circuits synthesised by the compiler. For example, the circuit for iteration is a set of asynchronous traces, which means that it processes one event per transition. Therefore, it cannot compose with the round-abstracted constants and it needs itself to be round-abstracted.

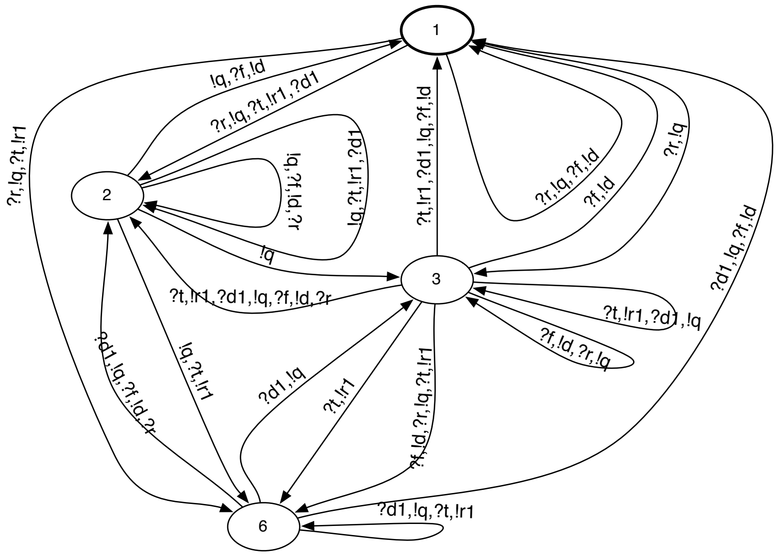


Fig. 4. Receptive round abstraction of iteration

An automaton for the original asynchronous representation of iteration is given in Fig. 3 (input events are marked with ? and outputs with !). The events in the iterator have the following interpretations: ?r: run the iterator; !q: request the evaluation of the guard; ?t, ?f: read the value of the guard; !r1: request the evaluation of the body; ?d1: receive the acknowledgement that the body completed execution; !d: report the completion of the iteration.

A receptive round abstraction is shown in Fig. 4. Because round abstraction gathers several transitions on a single new transition, it always reduce the number of states in the representation, which reduces the number of state bits in the implementation (flip-flops) and improves the latency. Consider for example what happens when the guard is false. In the asynchronous system, it takes four transitions to execute ($r.q.f.d$) whereas in the synchronous implementation there is a single transition for $\langle rqfd \rangle$, i.e. the latency in this case is zero.

6.2 Correctness

Failure of process compatibility can give rise to subtle bugs in synchronous implementations of GoS. Suppose that an implementer wants to reduce as much as possible the latency of (binary) memory locations, which following the game-semantic model are driven using the ports r (read), t (produce 1), f (produce 0), wt (write 1), wf (write 0), ok (acknowledge write). Singularity prevents multiple reads and multiple writes per round, but one read and one write per round could be implemented. A proper (asynchronous) memory cell trace such as $wf.ok.r.f.wt.ok.r.t$ could be presumably abstracted as $\langle wf, ok, r, f \rangle \langle wt, ok, r, t \rangle$. This is reasonable, and in fact, assignable variables in synchronous languages can be implemented like this (e.g. Esterel). However, such an implementation is incompatible with round abstraction because it breaks process compatibility and therefore partial round abstraction!

The reason is that an asynchronous program might generate local variable traces that are not consistent with stateful behaviour, but which are permutations of such traces; round abstraction may erroneously identify the two. An illegal trace such as $r.f.wf.ok.r.t.wt.ok$ can be also abstracted to $\langle wf, ok, r, f\rangle$ $\langle wt, ok, r, t\rangle$, which is the same as the abstraction of the legal trace above. At the level of the programming language, it means that programs x:=0; x:=1; if x=1 diverge and x:=0; if x=1 diverge; x:=1 could end up with the same implementation, which is obviously erroneous!

7 Conclusions

Representing asynchronous specifications as synchronous systems is generally useful, since synchronous systems are easier to implement as digital designs and asynchronous systems are more abstract and somewhat easier to specify. Naive implementations are unreasonably slow and expensive, but round abstraction is a general method for creating more efficient, lower latency circuits. In particular, we are mostly concerned with applicability to GoS.

In this paper, we give sufficient criteria to ensure the compositionality of both partial and total round abstractions. We restrict ourselves to round abstractions of well behaved asynchronous processes and, in the case of total round abstractions, we restrict ourselves to a certain class of round abstractions which we call "receptive". The classes of processes we are interested in are generalisations of game semantic models, therefore, they are motivated by pragmatic considerations. It remains to be seen whether these conditions can be further generalised or whether necessary conditions can be formulated.

Connections with Abramsky's Interaction Categories [17] should be examined, perhaps re-formulating round abstraction in that richer setting. Our simpler setting has the merit of introducing the basic concepts in a way that is directly applicable to pointer-free game models. However, the full power of Interaction Categories may offer a stronger platform to examine round abstraction in a principled way.

GoS will benefit directly from applying these results in the construction of correct compilers to synchronous circuits. However, the burden of proof for the compiler designer is still quite high. Verifying that the conditions of compatibility and safety are met is not trivial but essential, as Subsec. 6.2 shows. Our next step is to examine the compositional properties of compatibility and safety, aiming to ultimately identify a sub-category of asynchronous processes which can be correctly and compositionally round-abstracted. This sub-category will be the ideal framework to develop a correct compiler in a way that is guaranteed correct by construction.

Finally, our model uses a particular definition of safety (Def. 15) and the restrictions of *singularity* (Def. 4) and *seriality* (Def. 6) because, as explained, they are specific to our main intended application, modelling digital circuits. In this setting events are atomic and connectors are unable to buffer events. Eliminating these restrictions would lead to semantic models more suitable for higher-level languages and, quite possibly, a synchronous version of game semantics. This work is ongoing.

References

1. Milner, R.: Calculi for synchrony and asynchrony. Theor. Comput. Sci. 25, 267–310 (1983)
2. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
3. Halbwachs, N., Mandel, L.: Simulation and verification of asynchronous systems by means of a synchronous model. In: ACSD, pp. 3–14. IEEE Computer Society, Los Alamitos (2006)
4. Benveniste, A., Caillaud, B., Guernic, P.L.: From synchrony to asynchrony. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 162–177. Springer, Heidelberg (1999)
5. Alur, R., Henzinger, T.A.: Reactive modules. Formal Methods in System Design 15(1), 7–48 (1999)
6. Ghica, D.R.: Geometry of Synthesis: a structured approach to VLSI design. In: POPL, pp. 363–375 (2007)
7. Ghica, D.R.: Function interface models for hardware compilation: Types, signatures, protocols. CoRR abs/0907.0749 (2009)
8. Ghica, D.R.: Applications of game semantics: From software analysis to hardware synthesis. In: LICS, pp. 17–26 (2009)
9. Abramsky, S., Melliès, P.A.: Concurrent games and full completeness. In: LICS, pp. 431–442 (1999)
10. Ghica, D.R., Murawski, A.: Angelic semantics of fine-grained concurrency. Annals of Pure and Applied Logic 151(2-3), 89–114 (2008)
11. Ghica, D.R., Murawski, A.S., Ong, C.H.L.: Syntactic control of concurrency. Theor. Comput. Sci. 350(2-3), 234–251 (2006)
12. Berry, G., Gonthier, G.: The Esterel synchronous language: design, semantics and implementation. Technical Report 842, INRIA-Sophia Antiopolis (1988)
13. Chapiro, D.M.: Globally-asynchronous locally-synchronous systems. PhD thesis, Stanford Univ. (1984)
14. Gurd, J.R., Kirkham, C.C., Watson, I.: The Manchester prototype dataflow computer. Commun. ACM 28, 34–52 (1985)
15. Ghica, D.R., McCusker, G.: The regular-language semantics of second-order Idealized Algol. Theor. Comput. Sci. 309(1-3), 469–502 (2003)
16. Jifeng, H., Josephs, M.B., Hoare, C.A.R.: A theory of synchrony and asynchrony. In: Programming Concepts and Methods. Elsevier, Amsterdam (1990)
17. Abramsky, S.: Interaction categories. In: Theory and Formal Methods, pp. 57–69 (1993)
18. Abramsky, S.: Abstract interpretation, logical relations and Kan extensions. J. Log. Comput. 1(1), 5–40 (1990)
19. Nain, S., Vardi, M.Y.: Trace semantics is fully abstract. In: LICS, pp. 59–68 (2009)
20. Ghica, D.R., Smith, A.: Geometry of Synthesis II: From games to delay-insensitive circuits. In: MFPS XXVI (forthcoming, 2010)

A Linear Account of Session Types in the Pi Calculus

Marco Giunti[1] and Vasco T. Vasconcelos[2]

[1] Faculty of Planning, University IUAV of Venice
[2] LaSIGE, Faculty of Sciences, University of Lisbon

Abstract. We present a reconstruction of session types in a conventional pi calculus where types are qualified as linear or unrestricted. Linearly typed communication channels are guaranteed to occur in exactly one thread, possibly multiple times. We equip types with a constructor that denotes the two ends of a same communication channel. In order to assess the flexibility of the new type system, we provide three distinct encodings (from the linear lambda calculus, from the linear pi calculus, and from the pi calculus with polarized variables) into our system. For each language we present operational and typing correspondences, showing that our system effectively subsumes the linear pi calculus as well as foregoing works on session types.

1 Introduction

Session types allow a concise description of protocols by detailing the sequence of messages involved in each particular run of the protocol. Introduced for a dialect of the pi calculus [6,13], the concept has been transferred to different realms, including functional and object-oriented programming and operating systems; refer to [3] for a recent overview.

By way of motivation, consider a service allowing to create online petitions. Petition creators receive from the petition service a channel on which they provide the title of the petition, the petition text and the due date. After the initial setup, the exact same channel is ready to be distributed among the client's acquaintances to collect thousands of signatures, but not without the creator signing the petition first. The code for the creator can be written as follows,

$$\mathit{petitionOnline}(p).\overline{p}\,\mathit{title}.\overline{p}\,\mathit{description}.\overline{p}\,\mathit{dueDate}.\overline{p}\,\mathit{signature}.(\overline{a_1}\,p \mid \ldots \mid \overline{a_n}\,p)$$

where $x(y)$ denotes reading value y on channel x, $\overline{x}\,v$ denotes sending value v on channel x, and the vertical bar denotes parallel composition. Each of the acquaintances (not shown in the example), after reading p on channel a_i, can sign the petition and further distribute the channel at will.

The protocol for channel p can be concisely described by a type T of the form below, composed of an initial linear part that becomes shared (or unrestricted) in the later part.

$$\mathsf{lin}\,!\mathit{String}.\mathsf{lin}\,!\mathit{String}.\mathsf{lin}\,!\mathit{Date}.S \quad \text{where} \quad S = \mathsf{un}\,!\mathit{String}.S$$

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 432–446, 2010.

The final part is unrestricted because it is desiderable, but not absolutely necessary, that acquaintances (including the petition creator) sign the petition; conversely, the initial part is linear because petitions cannot be signed without first setting up the title, the description and the due date.

In the process above, each channel a_i forwards p at type S. Such a channel may be given the type $\mathsf{lin}!S.\mathsf{un}\,\mathsf{end}$, if we require that acquaintances eventually receive the petition channel; the continuation is $\mathsf{un}\,\mathsf{end}$ (the type of a channel on which no further interaction is possible) allowing channel a_i to be discarded thereafter. Concentrating on the type of *petitionOnline*, we see that petition creators need it a type $S_1 = \mathsf{un}?T.S_1$ so that they may create as many petitions (of type T) as required. It should be easy to see that the service itself sees the same channel at the dual type $S_2 = \mathsf{un}!T.S_2$. The whole system, composed by the service running in parallel with petition creators can be typed by reconciling the two *end point types* S_1 and S_2 in a single, unordered, *channel type* of the form (S_1, S_2).

The language of the pi calculus, when considered in conjunction with a type system with session types, is known to require a means to distinguish the two ends of a session channel (S_1 and S_2 above) in order to preserve type soundness [4,5,17]. Alternative solutions not requiring such a distinction rely on the restriction of channel passing to bound output. Such systems include the original formulation of delegation in session types [6] as well as more recent works [2,11].

Two approaches for distinguishing the ends of a channel are available in the literature: polarized channel variables [5], and form of channel double binder [14]. In the pi calculus with polarities the two ends of a channel x are distinguished by labelling each of its ends with a different label: x^+ and x^- denote the two ends of channel x. Given that from a given channel name one may find its two ends, one can restrict (the two ends of) a channel x with the usual pi calculus restriction operator $(\nu x)P$. Typing contexts, however accept two different entries for the same channel, one labelled with $+$, the other with $-$, as in the typing sequent below.

$$\Gamma, x^+ \colon S_1, x^- \colon S_2 \vdash \overline{x^+}\, v.P_1 \mid x^-(w).P_2$$

A variant of the above work, [14], uses distinct variables to describe the two ends of a same channel. In this case one cannot obtain the second end of a channel from the other end. It is restriction that puts together the two channel ends, by binding them together, as in $(\nu yz)P$. The assumptions in typing contexts are for simple variables, as in the example below where y and z denote the two ends of a same channel.

$$\Gamma, y \colon S_1, z \colon S_2 \vdash \overline{y}\, v.P_1 \mid z(w).P_2$$

The first work can be criticized for using non-conventional typing contexts, where typing information of a same channel x is split among two different entries, x^+ and x^-. The second work uses standard contexts but relies on a new scope restriction operator that binds two variables together. The goal of this work is to equip types with a constructor able to denote the two ends of a same channel. We then have the best of both worlds where we use the standard pi calculus (Milner et al. [10]) with standard typing sequents.

Syntax

$$\begin{array}{llr@{\qquad}llr}
b & ::= & \text{Booleans:} & P & ::= & \text{Processes:}\\
 & \mathsf{true} & \text{true} & & \overline{x}\,v.P & \text{output}\\
 & \mathsf{false} & \text{false} & & x(x).P & \text{input}\\
v & ::= & \text{Values:} & & P \mid P & \text{composition}\\
 & b & \text{boolean value} & & \mathsf{if}\,v\,\mathsf{then}\,P\,\mathsf{else}\,P & \text{conditional}\\
 & x & \text{variable} & & (\nu x)P & \text{restriction}\\
 & & & & *\,P & \text{replication}\\
 & & & & \mathbf{0} & \text{inaction}
\end{array}$$

Rules for structural congruence

$$P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad P \mid \mathbf{0} \equiv P \qquad *\,P \equiv P \mid *P$$
$$(\nu x)P \mid Q \equiv (\nu x)(P \mid Q) \qquad (\nu x)\mathbf{0} \equiv \mathbf{0} \qquad (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$$

Rules for reduction

$$\overline{x}\,v.P \mid x(y).Q \;\rightarrow\; P \mid Q[v/y] \qquad \text{[R-COM]}$$
$$\mathsf{if}\,\mathsf{true}\,\mathsf{then}\,P\,\mathsf{else}\,Q \;\rightarrow\; P \qquad \mathsf{if}\,\mathsf{false}\,\mathsf{then}\,P\,\mathsf{else}\,Q \;\rightarrow\; Q \qquad \text{[R-IFT] [R-IFF]}$$
$$\frac{P \rightarrow Q}{(\nu x)P \rightarrow (\nu x)Q} \qquad \frac{P \rightarrow Q}{P \mid R \rightarrow Q \mid R} \qquad \frac{P \equiv P' \quad P' \rightarrow Q' \quad Q' \equiv Q}{P \rightarrow Q}$$
$$\text{[R-RES] [R-PAR] [R-STRUCT]}$$

Fig. 1. Pi calculus: Syntax and operational semantics

We test the flexibility of our type system by embedding the pi calculus with polarities and session types [5] (hence the conventional pi calculus [10]). We do the same for the linear pi calculus [7], and for the linear (call by value) lambda calculus as in [16]. For each of these languages we prove an operational and a typing correspondence result. From the two first embeddings we learn that our type system is an extension of advanced type systems for pi calculi. The embedding of the linear lambda calculus crucially takes linearity into consideration generating code accordingly (replicated or non replicated) for shared and linear resources.

The outline of the paper is as follows. The next section recalls the pi calculus and introduces our type system. Then, the subsequent three sections present the embeddings of the three languages mentioned above: pi calculus with polarities, linear pi calculus and linear lambda calculus. The last section presents some related as well as future work.

2 Pi Calculus

This section introduces the pi-calculus, its syntax and semantics, as well as our type system. The syntax is in Figure 1. We rely on a set of variables, ranged over by x, y, z. Values include variables and the booleans true and false. For processes

q ::=	Qualifiers:
lin	linear
un	unrestricted
p ::=	Pre Types:
$?T.S$	receive
$!T.S$	send
end	termination
S ::=	End Point Types:
$q\ p$	qualified channel
a	type variable
$\mu a.S$	recursive type
T ::=	Types:
bool	boolean
S	end point
(S, S)	channel
Γ ::=	Contexts:
$\emptyset$	empty context
$\Gamma, x\colon T$	variable binding

Fig. 2. Pi calculus: Types and typing contexts

we have (synchronous, unary) output and input, in the forms $\overline{x}\,v.P$ and $x(y).P$, as well as a parallel composition, conditional, scope restriction, replication and the terminated process.

The binders for the language appear in parenthesis: x is bound in both $y(x).P$ and $(\nu x)P$. Free and bound variables in processes are defined accordingly, and so is alpha conversion, substitution of a variable x by a value v in a process P, denoted $P[v/x]$. We follow Barendregt's variable convention, requiring bound variables to be distinct from free variables in any mathematical context.

Structural congruence is the smallest relation on processes including the rules in the same figure. The first three rules say that parallel composition is commutative, associative and has $\mathbf{0}$ for neutral element. The last rule on the first line captures the essence of replication as an unbounded number of identical processes. The rules in the second line deal with scope restriction. The first, scope extrusion, allows the scope of x to encompass Q; due to variable convention, x bound in $(\nu x)P$, cannot be free in Q. The other two rules state that restricting over a terminated process has no effect, and allow exchanging the order of restrictions.

The reduction reduction is the smallest relation on processes including the rules in Figure 1. The [R-Com] rule communicates value v from an output prefixed one $\overline{x}\,v.P$ to an input prefixed process $x(y).Q$; the result is the parallel composition of the continuation processes, where the bound variable y is replaced by value v in the input process. The rules for the conditional are straightforward. The rules in the last line allow reduction to happen underneath scope restriction and parallel composition, and incorporate structural congruence into reduction.

The syntax of types is described in Figure 2. Types include the boolean type, end point types and channel types. The novelty with respect linear and session-based systems for the pi calculus is the introduction of a new type constructor to describe the two ends of a same channel, (S_1, S_2), where S_1 details the behaviour of one end, whereas S_2 details that of the other end. An end point type S can be a pre type qualified with lin or un, a recursive type or a type variable. Each qualifier in a type controls the number of times the channel can be used at that point: exactly once for lin; zero or more times for un. A pre type of the form

$!T.S$ describes a channel end able to send a value of type T and to proceed as prescribed by S. Similarly, pre type $?T.S$ describes a channel end able to receive a value of type T and continue as S. Pre type end describes a channel end on which no further interaction is possible. For recursive (end point) types we rely on a set of type variables, ranged over by a. Recursive types are required to be contractive, that is, containing no subexpression of the form $\mu a_1 \dots \mu a_n.a_1$.

Type equality is not syntatic. Instead, we define it as the equality of regular infinite trees obtained by the infinite unfolding of recursive types, *modulo pair commutation*. The formal definition, which we omit, is co-inductive. In this way we use types $(\mu a.\mathsf{lin}!\mathsf{bool}.\mathsf{lin}?\mathsf{bool}.a, \mathsf{un}\,\mathsf{end})$ and $(\mathsf{un}\,\mathsf{end}, \mathsf{lin}!\mathsf{bool}.\mu b.\mathsf{lin}?\mathsf{bool}.\mathsf{lin}!\mathsf{bool}.b)$ interchangeably, in any mathematical context. This allows us never to consider a type $\mu a.S$ explicitly (or a for that matter). Instead, we pick another type in the same equivalence class, namely $S[\mu a.S/a]$. If the result of the process turns out to start with a μ, we repeat the procedure. Unfolding is bound to terminate due to contractiveness. In other words, we take an equi-recursive view of types [12].

Type duality plays a central role in the theory of session types, ensuring that communication between the two ends of a channel proceeds smoothly. Intuitively, the dual of output is input and the dual of input is output. In particular if S_2 is dual of S_1, then $q?T.S_1$ is dual of $q!T.S_2$. Session type end is dual of itself. Rather than providing a co-inductive definition of duality, we start by defining a function from end-point channels into end-point channels as follows.

$$\overline{q\,?T.S} = q\,!T.\overline{S} \qquad \overline{q\,!T.S} = q\,?T.\overline{S} \qquad \overline{q\,\mathsf{end}} = q\,\mathsf{end} \qquad \overline{\mu a.S} = \mu a.\overline{S} \qquad \overline{a} = a$$

Then, to check that a given end point type S_1 is dual of another type S_2, we first build the dual of S_1 and then check that the thus obtained type is equivalent to S_2. For example, to show that type $\mu a.\mathsf{lin}?\mathsf{bool}.\mathsf{lin}!\mathsf{bool}.a$ is a dual of type $\mathsf{lin}!\mathsf{bool}.\mu b.\mathsf{lin}?\mathsf{bool}.\mathsf{lin}!\mathsf{bool}.b$, we build $\overline{\mu a.\mathsf{lin}?\mathsf{bool}.\mathsf{lin}!\mathsf{bool}.a} = \mu a.\mathsf{lin}!\mathsf{bool}.\mathsf{lin}?\mathsf{bool}.a$, and then show that $\mu a.\mathsf{lin}!\mathsf{bool}.\mathsf{lin}?\mathsf{bool}.a = \mathsf{lin}!\mathsf{bool}.\mu b.\mathsf{lin}?\mathsf{bool}.!\mathsf{bool}.b$. Qualifiers are important: S and $\overline{S}$ must be equally qualified so that a linear output process may find a linear input process to embark in reduction.

Contexts, or type environments, are inductively defined in Figure 2. In a context $\Gamma, x\colon T$ we assume that x does not occur in Γ; we also assume the various variable bindings in Γ to be unordered. We define predicate un to be true of a) the empty context, as well as of b) context $\Gamma, x\colon \mathsf{bool}$, context $\Gamma, x\colon \mathsf{un}\,p$, and context $\Gamma, x\colon (\mathsf{un}\,p_1, \mathsf{un}\,p_2)$, whenever $\mathsf{un}(\Gamma)$.

Typing relies on the context splitting operation described in Figure 3. It should be easy to understand: unrestricted types are copied into both contexts, linear types are placed in one of the two resulting contexts. The first four rules are standard [16], the last three rules are new to this work; the philosophy however remains the same. We omit three rules, duals to the last three, obtained by interchanging the end point types in the channel type (e.g., $(\mathsf{un}\,p_2, \mathsf{lin}\,p_1)$ in the last rule), for the effect can obtained by a suitable choice of the type in its equivalence class (recall that pair types are unordered).

Equipped with the notions of type duality, unrestricted contexts, and context splitting we are ready to introduce the typing rules in Figure 3. The first two

Context splitting rules

$$\emptyset = \emptyset \cdot \emptyset \qquad \frac{\Gamma = \Gamma_1 \cdot \Gamma_2 \qquad T = \mathsf{un}\, p \text{ or } (\mathsf{un}\, p_1, \mathsf{un}\, p_2)}{\Gamma, x\colon T = (\Gamma_1, x\colon T) \cdot (\Gamma_2, x\colon T)}$$

$$\frac{\Gamma = \Gamma_1 \cdot \Gamma_2 \qquad T = \mathsf{lin}\, p \text{ or } (\mathsf{lin}\, p_1, \mathsf{lin}\, p_2)}{\Gamma, x\colon T = (\Gamma_1, x\colon T) \cdot \Gamma_2} \qquad \frac{\Gamma = \Gamma_1 \cdot \Gamma_2 \qquad T = \mathsf{lin}\, p \text{ or } (\mathsf{lin}\, p_1, \mathsf{lin}\, p_2)}{\Gamma, x\colon T = \Gamma_1 \cdot (\Gamma_2, x\colon T)}$$

$$\frac{\Gamma = \Gamma_1 \cdot \Gamma_2}{\Gamma, x\colon (\mathsf{lin}\, p_1, \mathsf{lin}\, p_2) = (\Gamma_1, x\colon \mathsf{lin}\, p_1) \cdot (\Gamma_2, x\colon \mathsf{lin}\, p_2)}$$

$$\frac{\Gamma = \Gamma_1 \cdot \Gamma_2}{\Gamma, x\colon (\mathsf{lin}\, p_1, \mathsf{un}\, p_2) = (\Gamma_1, x\colon (\mathsf{lin}\, p_1, \mathsf{un}\, p_2)) \cdot (\Gamma_2, x\colon \mathsf{un}\, p_2)}$$

$$\frac{\Gamma = \Gamma_1 \cdot \Gamma_2}{\Gamma, x\colon (\mathsf{lin}\, p_1, \mathsf{un}\, p_2) = (\Gamma_1, x\colon \mathsf{un}\, p_2) \cdot (\Gamma_2, x\colon (\mathsf{lin}\, p_1, \mathsf{un}\, p_2))}$$

Typing rules for values

$$\frac{\mathsf{un}(\Gamma)}{\Gamma \vdash b\colon \mathsf{bool}} \qquad \frac{\mathsf{un}(\Gamma)}{\Gamma, x\colon T \vdash x\colon T} \qquad \frac{\Gamma \vdash v\colon (S, \mathsf{un}\, p)}{\Gamma \vdash v\colon S}$$

[T-Bool] [T-Var] [T-Strength]

Typing rules for processes

$$\frac{\mathsf{un}(\Gamma)}{\Gamma \vdash \mathbf{0}} \qquad \frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 \cdot \Gamma_2 \vdash P_1 \mid P_2} \qquad \frac{\Gamma \vdash P \qquad \mathsf{un}(\Gamma)}{\Gamma \vdash *P}$$

[T-Inact] [T-Par] [T-Repl]

$$\frac{\Gamma_1 \vdash v\colon \mathsf{bool} \qquad \Gamma_2 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 \cdot \Gamma_2 \vdash \mathsf{if}\, v\, \mathsf{then}\, P_1\, \mathsf{else}\, P_2} \qquad \frac{\Gamma, x\colon (S, \overline{S}) \vdash P}{\Gamma \vdash (\nu x)P}$$

[T-If] [T-Res]

$$\frac{\Gamma, x\colon S, y\colon T \vdash P \qquad (*)}{\Gamma, x\colon q?T.S \vdash x(y).P} \qquad \frac{\Gamma_1 \vdash v\colon T \qquad \Gamma_2, x\colon S \vdash P \qquad (**)}{\Gamma_1 \cdot (\Gamma_2, x\colon q\,!T.S) \vdash \overline{x}\, v.P}$$

[T-In],[T-Out]

$$\frac{\Gamma, x\colon (S, S'), y\colon T \vdash P \qquad (*)}{\Gamma, x\colon (q?T.S, S') \vdash x(y).P} \qquad \frac{\Gamma_1 \vdash v\colon T \qquad \Gamma_2, x\colon (S, S') \vdash P \qquad (**)}{\Gamma_1 \cdot (\Gamma_2, x\colon (q\,!T.S, S')) \vdash \overline{x}\, v.P}$$

[T-InC],[T-OutC]

$$(*)\ q = \mathsf{un} \Rightarrow q?T.S = S \qquad\qquad (**)\ q = \mathsf{un} \Rightarrow q!T.S = S$$

Fig. 3. Pi calculus: Typing

typing rules for values are standard. Rule [T-Strength] is central to our system with channels described as pairs of types; we discuss it after introducing the remaining typing rules.

For processes, rule [T-Inact] says that the terminated process can only be typed in an unrestricted context, ensuring that linear channels are given a chance to be consumed. Rule [T-Par] uses context splitting to partition linearly typed variables between the two processes: the incoming context is split into Γ_1 and Γ_2, and we use the former to type check process P_1 and the latter to type check process P_2. Rule [T-Repl] for replication requires the typing context not to contain linear values, for P may be used an unrestricted number of types. Rule [T-If] for the conditional process splits the incoming context in two parts: one used to check the condition, the other to check both branches. The same context for the two branches is justified by the fact that only one of P_1 or P_2 will be executed. Rule [T-Res] allows restricting channels whose end points are dual, making sure that communication on the channel happens according to the plan. Allowing to restrict a end point type S type would not break type preservation, Theorem 1, but we believe that such an alternative rule does not fit well in a linear system, where we expect linear channels to be given an opportunity to be consumed. Even though unrestricted end point types cannot be directly restricted, we can show that, for each derivation of $\Gamma, x\colon S \vdash P$, there is a derivation of $\Gamma, x\colon (S, \mathsf{un}\, p) \vdash P$, thus allowing to apply scope restriction to an otherwise unrestricted channel end.

We have two rules for input, [T-In] and [T-InC], depending on the type for channel x in the context. Rule [T-In] deals with end point types. If x is typed with $q\,?T.S$, we know that the bound variable y is of type T, and we type check P under the extra assumption $y\colon T$. Equally important is the fact that the continuation uses channel x at continuation type S, that is, process $x(y).P$ uses channel x at type $q\,?T.S$ whereas P may use the *same* channel this time at type S. Finally, unrestricted channels, given that they may be shared, must retain their behavior throughout computation, hence the side condition. A solution to the equation in the side condition is $\mu a?T.a$ for a not in T, which we abbreviate to $*?T$ (and similarly for output). Rule [T-InC] follows the same pattern, consuming one end point and keeping the other unchanged. Similarly to input, we have two rules for output. Rule [T-Out], splits the context in two parts, one to check v and the other to check continuation P. Notice that the context in the conclusion, $\Gamma_1 \cdot (\Gamma_2, x\colon q\,!T.S)$ allows to type process $\overline{x}\,x$ with a context $x\colon S$ with type S such that $S = \mathsf{un}!S.S$.

Rule [T-Strength] allows for a fine grained control of the channel ends of a given channel. A process holding the two ends of a given channel x, say $(*!\mathsf{bool}, *?\mathsf{bool})$, may pass the output capability only by using [T-Var] followed by [T-Strength] to obtain $\Gamma, x\colon (*!\mathsf{bool}, *?\mathsf{bool}) \vdash x\colon *!\mathsf{bool}$ and then compose with rule [T-Out] or [T-OutC] in a process of the form $\overline{y}\,x.P$. The rule is also fundamental in establishing the main result of this section.

To lighten the syntax in examples, we omit all unrestricted qualifiers and only annotate linear types. We also omit the trailing $\mathsf{un}\,\mathsf{end}$ in types, as well as the trailing $\mathbf{0}$ in processes. As an example, consider the type $?(\mathsf{lin}!\mathsf{bool}).S$ of an unrestricted channel that receives a linear channel capable of outputting a boolean value. The following sequent is easy to establish,

$$x\colon ?(\mathsf{lin}!\mathsf{bool}).S \vdash x(z).\overline{z}\,\mathsf{true} \mid x(w).\overline{w}\,\mathsf{false}$$

but only for an appropriate type S. Reading rule [T-In], we realize that S must be equivalent to $?(\mathsf{lin}!\mathsf{bool}).S$, that is S must be (equivalent to) $\mu a.?(\mathsf{lin}!\mathsf{bool}).a$, abbreviated to $*?(\mathsf{lin}!\mathsf{bool})$. Continuing with the example, if P is the above process, then $P \mid (\nu y)\overline{x}\,y$ is not typable, for the linear input capability of channel y is never exercised. But $P \mid (\nu y)(\overline{x}\,y \mid y(u))$ is typable under context $x\colon (*?(\mathsf{lin}!\mathsf{bool}), *!(\mathsf{lin}?\mathsf{bool}))$.

Given that a type $(\mathsf{un}\,p_1, \mathsf{lin}\,p_2)$ cannot possibly be restricted in a process (cf. rule [T-Res]), the reader may wonder why we consider them at all. It turns out that free channel output may lead to situations where a thread holds the two ends of a same channel [17]. For instance, process $\overline{z}\,x \mid z(w).w(y).\overline{x}\,\mathsf{true}$, typable under context $z\colon (*?S, *!S), x\colon (S, \overline{S})$ with $S = \mathsf{lin}?\mathsf{bool}$, reduces to process $P = x(y).\overline{x}\,\mathsf{true}$, which we want to type under the same context. By applying rule [T-InC] to P we obtain a judgment with a un-lin type, namely $z\colon (*?S, *!S), x\colon (\mathsf{end}, \mathsf{lin}!\mathsf{bool}) \vdash \overline{x}\,\mathsf{true}$. A further application of rule [T-OutC] gets rid of the un-lin type, yielding $z\colon (*?S, *!S), x\colon (\mathsf{end}, \mathsf{end}) \vdash \mathbf{0}$ typable under rule [T-Inact].

We conclude the section with the main result of our system. Reduction preserves typability only for a certain kind of contexts. To understand why reduction does not preserve typability in the presence of arbitrary contexts, take for P the process $x(z).\mathsf{if}\, z\, \mathsf{then}\, \mathbf{0}\, \mathsf{else}\, \mathbf{0} \mid (\nu y)\overline{x}\,y$. We can easily see that P is typable under the (non balanced) context $x\colon (\mathsf{lin}!\mathsf{end}.\mathsf{end}, \mathsf{lin}?\mathsf{bool}.\mathsf{end})$. But P reduces to process $(\nu y)\mathsf{if}\, y\, \mathsf{then}\, \mathbf{0}\, \mathsf{else}\, \mathbf{0}$ which is not typable. The whole purpose of balancing is to make sure that the type of y in the output is that of z in the input.

We define predicate *balanced* to be true of a) the empty context, and b) context $\Gamma, x\colon \mathsf{bool}$ and context $\Gamma, x\colon (S, \overline{S})$ whenever Γ is balanced.

Theorem 1 (Type Preservation). *If $\Gamma_1 \vdash P_1$ with Γ_1 balanced and $P_1 \to P_2$, then $\Gamma_2 \vdash P_2$ with Γ_2 balanced.*

Proof (Sketch). Albeit standard, the proof is quite long due to the combinatorics introduced by (the various rules in) context splitting, the lin-un qualifiers and the four rules for input and for output. We rely on several standard auxiliary results, including unrestricted context weakening, context strengthening, a substitution lemma (stating that if $\Gamma_1 \vdash v\colon T$ and $\Gamma_2, x\colon T \vdash P$ and $\Gamma_1 \cdot \Gamma_2$ is defined then $\Gamma_1 \cdot \Gamma_2 \vdash P[v/x]$), and balanced context preservation for structural congruence. In order to proceed by induction on the derivation of the reduction step when [T-Par] is the last rule, we need a stronger statement that details the relation between Γ_1 and Γ_2, namely $\Gamma_2 = \Gamma_1$ or $\Gamma_1 = \Gamma, x\colon (q?T.S, q!T.\overline{S})$ and $\Gamma_2 = \Gamma, x\colon (S, \overline{S})$.

3 Embedding the Pi Calculus with Polarities

This section shows that our type system embeds the polarity system introduced by Gay and Hole [5]. Since Gay and Hole show that the pi calculus with polarities embeds the simply typed pi calculus; by transitivity our language embeds the simply typed pi calculus as well.

In Figure 4 we present the branch-select free fragment of the pi calculus with polarities. Variables may be *polarized*, occurring in processes as well as in typing contexts as x^+ or x^- or simply as x. We write x^p for a general polarized name, where p represents an optional polarity. Duality on polarities, written $\overline{p}$ exchanges $+$ and $-$. The new constructors of the language, input and output, are in Figure 4; the remaining are taken from Figure 1; the syntactic category for values in Figure 1 does not contribute to the language.

New syntactic forms

$$\begin{array}{lllr} P & ::= & \dots & \text{Processes:} \\ & & \overline{x^p}\,x^p.P & \text{output} \\ & & x^p(x).P & \text{input} \\ T & ::= & & \text{Types:} \\ & & \hat{\ }T & \text{standard channel} \\ & & S & \text{session channel} \\ & & a & \text{type variable} \end{array} \qquad \begin{array}{lllr} & & \mu a.S & \text{recursive type} \\ S & ::= & & \text{Session types:} \\ & & \mathsf{end} & \text{termination} \\ & & ?T.S & \text{receive} \\ & & !T.S & \text{send} \\ & & a & \text{type variable} \\ & & \mu a.S & \text{recursive type} \end{array}$$

New reduction rules

$$\overline{x^p}\,z^q.P \mid x^{\overline{p}}(y).Q \to_{\mathsf{p}} P \mid Q[z^q/y] \qquad \text{[R-Com]}$$

Context updating

$$\begin{array}{rl} \Gamma + x^p\colon S = \Gamma, x^p\colon S & \text{if } x^p, x \notin \mathrm{dom}(\Gamma) \text{ and } S = ?T.S, !T.S, \mathsf{end} \\ \Gamma + x\colon T = \Gamma, x\colon T & \text{if } x, x^+, x^- \notin \mathrm{dom}(\Gamma) \\ \Gamma, x\colon T + x\colon T = \Gamma, x\colon T & \text{if } T = \hat{\ }T, \mathsf{bool} \end{array}$$

Typing rules

$$\frac{\Gamma \text{ completed}}{\Gamma \vdash_{\mathsf{p}} \mathbf{0}} \qquad \frac{\Gamma \vdash_{\mathsf{p}} P \qquad \Gamma \text{ unlimited}}{\Gamma \vdash_{\mathsf{p}} *P} \qquad \frac{\Gamma, x\colon \hat{\ }T \vdash_{\mathsf{p}} P}{\Gamma \vdash_{\mathsf{p}} (\nu x)P} \qquad \frac{\Gamma, x^+\colon S, x^-\colon \overline{S} \vdash_{\mathsf{p}} P}{\Gamma \vdash_{\mathsf{p}} (\nu x)P}$$

[T-Inact] [T-Repl] [T-New] [T-NewS]

$$\frac{\Gamma_1 \vdash_{\mathsf{p}} P \qquad \Gamma_2 \vdash_{\mathsf{p}} Q}{\Gamma_1 + \Gamma_2 \vdash_{\mathsf{p}} P \mid Q} \qquad \frac{\Gamma, x\colon \hat{\ }T, y\colon T \vdash_{\mathsf{p}} P}{\Gamma, x\colon \hat{\ }T \vdash_{\mathsf{p}} x(y).P} \qquad \frac{\Gamma, x^p\colon S, y\colon T \vdash_{\mathsf{p}} P}{\Gamma, x^p\colon ?T.S \vdash_{\mathsf{p}} x^p(y).P}$$

[T-Par] [T-In] [T-InS]

$$\frac{\Gamma, x\colon \hat{\ }T \vdash_{\mathsf{p}} P}{(\Gamma, x\colon \hat{\ }T) + y^q\colon T \vdash_{\mathsf{p}} \overline{x}\,y^q.P} \qquad \frac{\Gamma, x^p\colon S \vdash_{\mathsf{p}} P}{(\Gamma, x^p\colon !T.S) + y^q\colon T \vdash_{\mathsf{p}} \overline{x^p}\,y^q.P}$$

[T-Out] [T-OutS]

Fig. 4. Pi calculus with polarities

The reduction relation, denoted by $\to_{\mathsf{p}}$, is defined inductively by the rules in Figure 1 with rule [R-Com] replaced by that in Figure 4. From the above description it should be obvious that the two languages differ in the (optional) polarity annotation on (non-bound occurrences of) variables. We define an erase function that removes from a polarized processes all occurrences of $+$ and $-$, to yield a process generated by the grammar in Figure 1. There is an obvious operational correspondence between the two languages, stated in Theorem 2. The converse is clearly not true. Take for P the polarized process $\overline{x^+} \mid x^+()$. Then $\mathrm{erase}(P) = \overline{x} \mid x()$ reduces while P does not.

The language of types includes a distinct category S for (linear) session types. Since we restrict our language to the branch-select free fragment of [5], we ignore subtyping. Duality is defined as in Section 2, with the appropriate changes which amount erasing the qualifiers. Typing contexts now gather assumptions on polarized variables, in addition to simple variables as before. There is however one restriction on the variables occurring in a context: x and x^+ (or x^-) cannot occur simultaneously in a given context Γ, even though x^+ and x^- may. New assumptions are added to contexts by means of an update operation $+$, defined in Figure 4. Context updating is different from splitting (in Figure 3) on what concerns unrestricted types: $\emptyset + (x\colon \hat{}T)$ is defined, whereas $\emptyset \cdot (x\colon \mathsf{un}\, p)$ is not. We say that a context is *unlimited* if it contains no session types, and is *completed* if every session in it is end.

The typing relation is inductively defined by the rules in Figure 4. Rule [T-NewS] requires the types for the two channel end points to be of dual types; contrast with rule [T-Res] in Figure 3: our system merges the two end points in a single variable and requires the two components of the channel type to be of dual types. Rules [T-In] and [T-InS] in Figure 4 have their counterpart in rules [T-In] and [T-InC] in Figure 3. The choice here is not based on whether the type for the input channel is an end point or a channel type but rather on whether the qualifier is linear or unrestricted. The same can be said of rules [T-Out] and [T-OutS].

From the above description it should be obvious that the two systems are quite close to each other. In order to define the typing correspondence we need to translate types and contexts for the polarized language (as in Figure 4) to those in our language (Figure 1). The definition is as follows; recall from Section 2 that we use $*?T$ as an abbreviation for $\mu a.?T.a$, for some a not in T. To translate typing contexts we assume that if both x^+ and x^- are in Γ then they occur in contiguous positions (and in this order). The translation of typing contexts is as follows, where the rules must be tried in the given order; the first rule for mapping non-empty contexts is for polarized pairs while the second rule is for single entries.

$$\begin{array}{rlrlrl}
\llbracket \hat{}T \rrbracket &= (*?\llbracket T \rrbracket, *!\llbracket T \rrbracket) & \llbracket \mathsf{end} \rrbracket &= \mathsf{un\,end} & \llbracket \emptyset \rrbracket &= \emptyset \\
\llbracket ?T.S \rrbracket &= \mathsf{lin}?\llbracket T \rrbracket.\llbracket S \rrbracket & \llbracket a \rrbracket &= a & \llbracket \Gamma, x^+\colon S, x^-\colon S' \rrbracket &= \llbracket \Gamma \rrbracket, x\colon (\llbracket S \rrbracket, \llbracket S' \rrbracket) \\
\llbracket !T.S \rrbracket &= \mathsf{lin}!\llbracket T \rrbracket.\llbracket S \rrbracket & \llbracket \mu a.S \rrbracket &= \mu a.\llbracket S \rrbracket & \llbracket \Gamma, x^p\colon T \rrbracket &= \llbracket \Gamma \rrbracket, x\colon \llbracket T \rrbracket \\
 & & \llbracket \mu a.T \rrbracket &= \mu a.\llbracket T \rrbracket & &
\end{array}$$

We are now in a position to state the main result of this section.

Theorem 2 (Polarity-Pi To Pi Correspondence).

1. *If* $\Gamma \vdash_{\mathsf{p}} P$ *then* $\llbracket \Gamma \rrbracket \vdash \mathrm{erase}(P)$.
2. *If* $P \rightarrow_{\mathsf{p}} Q$, *then* $\mathrm{erase}(P) \rightarrow \mathrm{erase}(Q)$.

4 Embedding the Linear Pi Calculus

In this section we analyse (a synchronous variant of) the linear pi calculus [7] and provide a typing-preserving encoding into our system.

New syntactic forms

$$\begin{array}{llrcllr}
c & ::= & \text{Capabilities:} & \qquad & T & ::= & \text{Types:} \\
 & \mathsf{i} & \text{input} & & & q\,c\,T & \text{channel} \\
 & \mathsf{o} & \text{output} & & & \mathsf{bool} & \text{boolean} \\
 & \mathsf{io} & \text{input and output} & & & &
\end{array}$$

Combination of types

$$\mathsf{bool} + \mathsf{bool} = \mathsf{bool} \qquad \mathsf{un}\,c_1\,T + \mathsf{un}\,c_2\,T = \mathsf{un}\,(c_1 \cup c_2)\,T \qquad \mathsf{lin}\,\mathsf{i}\,T + \mathsf{lin}\,\mathsf{o}\,T = \mathsf{lin}\,\mathsf{io}\ T$$

Combination of contexts

$$(\Gamma_1 + \Gamma_2)(x) = \begin{cases} \Gamma_1(x) + \Gamma_2(x) & \text{if } x \in \mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) \\ \Gamma_1(x) & \text{if } x \in \mathrm{dom}(\Gamma_1) \text{ and } x \notin \mathrm{dom}(\Gamma_2) \\ \Gamma_2(x) & \text{if } x \in \mathrm{dom}(\Gamma_2) \text{ and } x \notin \mathrm{dom}(\Gamma_1) \end{cases}$$

Typing rules for processes

$$\frac{\Gamma \vdash_{\mathsf{l}} P_1 \qquad \Gamma \vdash_{\mathsf{l}} P_2}{\Gamma + v : \mathsf{bool} \vdash_{\mathsf{l}} \mathsf{if}\, v\, \mathsf{then}\, P_1\, \mathsf{else}\, P_2} \qquad \frac{\Gamma, x \colon q\,\mathsf{io}\ T \vdash_{\mathsf{l}} P}{\Gamma \vdash_{\mathsf{l}} (\nu x)P} \qquad \text{[T-If] [T-Res]}$$

$$\frac{\Gamma, y \colon T \vdash_{\mathsf{l}} P}{\Gamma + x \colon q\,\mathsf{i}T \vdash_{\mathsf{l}} x(y).P} \qquad \frac{\Gamma \vdash_{\mathsf{l}} P}{\Gamma + x \colon q\,\mathsf{o}T + v \colon T \vdash_{\mathsf{l}} \overline{x}\,v.P} \qquad \text{[T-In] [T-Out]}$$

Fig. 5. Linear pi calculus

The syntax of linear pi processes and the reduction relation are described in Figure 1. Figure 5 defines the syntax of types and the typing rules for processes. Types have now the form $q\,c\,T$ where c is a capability formally defined as one of the following sets.

$$\mathsf{i} = \{\mathsf{i}\} \qquad\qquad \mathsf{o} = \{\mathsf{o}\} \qquad\qquad \mathsf{io}\ = \{\mathsf{i}, \mathsf{o}\}$$

The linear discipline is imposed by way of a $+$ combination operation over types, defined in Figure 5. The operator is extended point-wise to typing contexts. Notice that context combination is different from the context splitting operation defined in Figure 3 when in the presence of unrestricted types: context splitting does not allow composing $(\Gamma_1, x \colon \mathsf{un}\,c\,T)$ with Γ_2 whenever $x \notin \mathrm{dom}(\Gamma_2)$ or when $\Gamma_2(x) \neq \mathsf{un}\,c\,T$.

The typing system for the linear pi-calculus is defined by the rules in Figure 3 together with rules [T-Inact], [T-Repl] and [T-Par] in Figure 4. Rule [T-Out] is an adaptation of that in [7] to the synchronous setting: we let the continuation be typed with context Γ while in the original paper the premise to the rule is $\mathsf{un}(\Gamma)$ since the (absent) continuation behaves as $\mathbf{0}$. We also adapt rule [T-Res] to require that the restricted channel uses both capabilities; the original system allows processes of the form $(\nu x)\overline{x}\,\mathsf{true}$ to be typed by assigning to channel x type $\mathsf{lin}\,\mathsf{o}\,\mathsf{bool}$; cf. discussion around rule [T-Res] in Section 2.

The compositional encoding of linear types is defined below and is useful to understand the reconstruction of session types introduced in Section 2. A linear input (output) type is embedded as a linear input (output) type whose continuation is $\mathsf{un\,end}$, meaning that the continuation process cannot further use the channel. Unrestricted input (output) types are mapped into unrestricted recursive input (output) types. For instance, the type $\mathsf{lin\ i}(\mathsf{lin\ io}\ (\mathsf{un\ io\ bool}))$ is mapped into the type $\mathsf{lin}\ ?(\mathsf{lin}\ !T.\mathsf{un\,end}, \mathsf{lin}\ ?T.\mathsf{un\,end}).\mathsf{un\,end}$ where $T = (*!\mathsf{bool}, *?\mathsf{bool})$.

$$\begin{aligned}
[\![\mathsf{lin\ i}T]\!] &= \mathsf{lin}?[\![T]\!].\mathsf{un\,end} & [\![\mathsf{lin\ o}T]\!] &= \mathsf{lin}![\![T]\!].\mathsf{un\,end}\\
[\![\mathsf{un\ i}T]\!] &= *?[\![T]\!] & [\![\mathsf{un\ o}T]\!] &= *![\![T]\!]\\
[\![q\ \mathsf{io}\ T]\!] &= ([\![q\ \mathsf{i}T]\!], [\![q\ \mathsf{o}T]\!]) & [\![\mathsf{bool}]\!] &= \mathsf{bool}
\end{aligned}$$

The main result of this section establishes the correspondence between the two systems.

Theorem 3 (Linear-Pi To Pi Correspondence). *If $\Gamma \vdash_{\mathsf{l}} P$ then $[\![\Gamma]\!] \vdash P$.*

5 Embedding the Linear Lambda Calculus

This section shows that the call-by-value linear lambda calculus can be faithfully encoded in our language. We follow the presentation of Walker [16], except that we use an implicitly typed language.

The syntax of the language is in Figure 6; we rely on the set of variables introduced in Section 2 for the pi calculus; the missing non-terminal symbols, q, b and so on, are in Figure 1. Values are qualified, linear or unrestricted, and include boolean values and abstractions. Terms are variables, values, and applications and are evaluated in an abstract machine with an explicit store. The store is a sequence of variable-value pairs, treated as a map from variables into values. To simplify the presentation of the evaluation relation, we use an auxiliary function, $S \overset{q}{\sim} x$ that deallocates the value associated with variable x in S when the qualifier q is lin, and leaves S unchanged otherwise. The evaluation reduction copies values into the store, associating them with a fresh variable (rule [E-VAL]). For function application the value associated with the function is looked upon in the store; if linear it is then deallocated (rule [E-APP]). The remaining two rules implement the call-by-value strategy. We denote by $\rightarrow_\lambda$ the reduction relation in Figure 6.

For typing, rule [T-VAR] is that of the pi-calculus (Figure 3). Rule [T-BOOL] contrasts with its homonymous in Figure 3 in that values in the linear lambda calculus are qualified, the type of a value inheriting the qualifier of the value. The remaining two rules, for abstraction and application, are standard in the linear lambda calculus; notice the $q(\Gamma)$ in rule [T-ABS] requiring an unrestricted function to contain only unrestricted free variables ($\mathsf{un}(\Gamma)$ is defined in Section 2; $\mathsf{lin}(\Gamma)$ is true).

For the translation we rely on a polyadic variant of the pi language, allowing channels to carry an arbitrary (but fixed) number of values. The extension is

Syntax

$$\begin{array}{lllr@{\qquad}lllr}
v & ::= & & \text{Values:} & p & ::= & & \text{Pretypes:} \\
 & & q\ b & \text{boolean} & & & \mathsf{bool} & \text{boolean} \\
 & & q\ \lambda x.M & \text{abstraction} & & & T \to T & \text{function} \\
M & ::= & & \text{Terms:} & T & ::= & & \text{Types:} \\
 & & x & \text{variable} & & & q\ p & \text{qualified pretype} \\
 & & v & \text{value} & S & ::= & & \text{Stores:} \\
 & & MM & \text{application} & & & \emptyset & \text{empty store} \\
 & & & & & & S, x \mapsto v & \text{store binding}
\end{array}$$

Store deallocation

$$(S_1, x \mapsto v, S_2) \overset{\mathsf{lin}}{\sim} x = S1, S2 \qquad\qquad S \overset{\mathsf{un}}{\sim} x = S$$

Evaluation

$$(S; v) \to_\lambda (S, x \mapsto v; x) \qquad \frac{S(x_1) = q\lambda y.M}{(S; x_1 x_2) \to_\lambda (S \overset{q}{\sim} x_1; M[x_2/y])} \qquad \text{[E-VAL] [E-APP]}$$

$$\frac{(S; M_1) \to_\lambda (S'; M_1')}{(S; M_1 M_2) \to_\lambda (S'; M_1' M_2)} \qquad \frac{(S; M) \to_\lambda (S'; M')}{(S; xM) \to_\lambda (S'; xM')} \qquad \text{[E-FUN] [E-ARG]}$$

Typing

$$\frac{\mathsf{un}(\Gamma)}{\Gamma, x\colon T \vdash_\lambda x\colon T} \qquad \frac{\mathsf{un}(\Gamma)}{\Gamma \vdash_\lambda q\,b\colon q\,\mathsf{bool}} \qquad \text{[T-VAR] [T-BOOL]}$$

$$\frac{\Gamma, x\colon T_1 \vdash_\lambda M\colon T_2 \qquad q(\Gamma)}{\Gamma \vdash_\lambda q\lambda x.M\colon q\,T_1 \to T_2} \qquad \frac{\Gamma_1 \vdash_\lambda M_1\colon q\,T_1 \to T_2 \qquad \Gamma_2 \vdash_\lambda M_2\colon T_1}{\Gamma_1 \cdot \Gamma_2 \vdash_\lambda M_1 M_2\colon T_2}$$

[T-ABS] [T-APP]

Fig. 6. Linear lambda calculus

straightforward to incorporate: for processes we need polyadic output and input, $x(\vec{x}).P$ and $\overline{x}\,\vec{v}.P$; for types (pre types, rather) we need $?\langle\vec{T}\rangle.S$ and $!\langle\vec{T}\rangle.S$. The extension can be incorporated in the base theory or added as an encoding [14].

On what concerns the translation of types below, the interesting cases are the two forms of arrow types. An unrestricted $T_1 \to T_2$ type is translated as a pair of types: the $*?X$ part caters for the resource (the function proper) and the $*!X$ for its clients. Channels describing functions carry a pair X of values: the first element in the pair is the argument to the function ($[\![T_1]\!]$ in X); the second is a channel that will convey the result and that will be used exactly once (a linear channel of type $\mathsf{lin}![\![T_2]\!].\mathsf{un\,end}$). The type for linear resources is similar, only that they are linear, rather than unrestricted. The translation of terms follows that of Milner [9] with two exceptions. On the one hand, the value qualifiers are taken into consideration in the translation: a linear value is translated into a simple output (in the case of a boolean value) or a simple input (in the case of an abstraction); only for unrestricted values replication is used. On the other hand, applications of the form xM are partially evaluated, allowing for a simple operational correspondence (cf. [15]).

$$\begin{aligned}
\llbracket q\ \mathsf{bool} \rrbracket &= q\ \mathsf{bool} \\
\llbracket \mathsf{un}\ T_1 \to T_2 \rrbracket &= (*?X, *!X) \\
\llbracket \mathsf{lin}\ T_1 \to T_2 \rrbracket &= (\mathsf{lin}?X.\mathsf{un\,end}, \mathsf{lin}!X.\mathsf{un\,end}) \\
\text{where } X &= \langle \llbracket T_1 \rrbracket, \mathsf{lin}!\llbracket T_2 \rrbracket.\mathsf{un\,end} \rangle \\
\llbracket \emptyset \rrbracket &= \mathbf{0} \\
\llbracket S, x \mapsto v \rrbracket &= \llbracket S \rrbracket \mid \llbracket x \mapsto v \rrbracket \\
\llbracket x \mapsto q\ b \rrbracket &= \llbracket q \rrbracket \overline{x}\, b \\
\llbracket x \mapsto q\lambda y M \rrbracket &= \llbracket q \rrbracket x(yp).\llbracket M \rrbracket_p
\end{aligned}$$

$$\begin{aligned}
\llbracket x \rrbracket_p &= \overline{p}\, x \\
\llbracket v \rrbracket_p &= (\nu x)(\llbracket x \mapsto v \rrbracket \mid \overline{p}\, x) \\
\llbracket xM \rrbracket_p &= (\nu r)(\llbracket M \rrbracket_r \mid r(y).\overline{x}\, yp) \\
\llbracket MN \rrbracket_p &= (\nu s)(\llbracket M \rrbracket_s \mid s(x).(\nu r)(\\
&\qquad \llbracket N \rrbracket_r \mid r(y).\overline{x}\, yp)) \\
\llbracket S; M \rrbracket &= (\nu\, \mathrm{dom}(S))(\llbracket S \rrbracket \mid \llbracket M \rrbracket) \\
\llbracket \mathsf{un} \rrbracket &= * \\
\llbracket \mathsf{lin} \rrbracket &= \text{the empty string}
\end{aligned}$$

We are now in a position to state the main result of this section. Let $\to^*$ be the reflexive and transitive closure of the reduction relation $\to$ defined in Figure 1.

Theorem 4 (Linear-Lambda to Pi Correspondence)

1. *If* $\Gamma \vdash_\lambda M : T$, *then* $\llbracket \Gamma \rrbracket, p \colon \mathsf{lin}!\llbracket T \rrbracket.\mathsf{un\ end} \vdash \llbracket M \rrbracket_p$.
2. *If* $(S; M) \to_\lambda (S'; M')$, *then* $\llbracket S; M \rrbracket \to^* \llbracket S'; M' \rrbracket$.

6 Conclusion

As mentioned in the introduction, the pi calculus equipped with a polarity-based typing system [5] and the (double binder) pi calculus equipped with a conventional typing system [14] are the works closest to ours. Here we try to obtain the same results, relying on the traditional pi calculus equipped with a conventional type system. Towards this end we introduce an unordered pair constructor denoting, at type level, the two ends of a same channel. In order to distinguish linear from unrestricted variables we use type qualifiers applied to pre types, inspired by Walker's presentation of substructural type systems [16]. Caires and Pfenning take a different approach, closely adhering to linear logic, treating all variables as linear and using exponential "!" to describe shared resources [1].

Algorithmic type checking is left for further work; the language with double binders [14] is equipped with such a system. Gay and Hole address the problem of polarity inference for closed processes under certain restrictions [5]. Particularly promising is the F$^\circ$ ("F-pop") system by Maruzak et al. [8], where a kinding system, instead of type qualifiers, simplifies the use of linearity in functional programming languages, including a novel form of subtyping between linear and unrestricted kinds, which we would like to explore.

Acknowledgments. This work was supported by FCT/MCTES via projects PTDC/EIA–CCO/105359/2008 and CMU–PT/NGN44–2009–12. The authors would like thank Simon Gay and Nils Gesbert for insightful comments and the anonymous referees for constructive criticisms and detailed comments.

References

1. Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 222–236. Springer, Heidelberg (2010)
2. Castagna, G., Dezani-Ciancaglini, M., Giachino, E., Padovani, L.: Foundations of session types. In: PPDP, pp. 219–230. ACM, New York (2009)
3. Dezani-Ciancaglini, M., de'Liguoro, U.: Sessions and session types: An overview. In: Laneve, C. (ed.) WS-FM 2010. LNCS, vol. 6194, pp. 1–28. Springer, Heidelberg (2010)
4. Dezani-Ciancaglini, M., Drossopoulou, S., Mostrous, D., Yoshida, N.: Objects and session types. Information and Computation 207, 595–641 (2009)
5. Gay, S.J., Hole, M.J.: Subtyping for session types in the pi calculus. Acta Informatica 42(2/3), 191–225 (2005)
6. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998)
7. Kobayashi, N., Pierce, B.C., Turner, D.N.: Linearity and the pi-calculus. ACM Transactions on Programming Languages and Systems 21, 914–947 (1999)
8. Mazurak, K., Zhao, J., Zdancewic, S.: Lightweight linear types in system F^o. In: TLDI, pp. 77–88. ACM, New York (2010)
9. Milner, R.: Functions as processes. Mathematical Structures in Computer Science 2(2), 119–141 (1992)
10. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, part I/II. Journal of Information and Computation 100, 1–77 (1992)
11. Padovani, L.: Session types at the mirror. EPTCS 12, 71–86 (2009)
12. Pierce, B.C.: Types and Programming Languages. MIT Press, Cambridge (2002)
13. Takeuchi, K., Honda, K., Kubo, M.: An Interaction-based Language and its Typing System. In: Halatsis, C., Philokyprou, G., Maritsas, D., Theodoridis, S. (eds.) PARLE 1994. LNCS, vol. 817, pp. 398–413. Springer, Heidelberg (1994)
14. Vasconcelos, V.T.: Fundamentals of Session Types. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009. LNCS, vol. 5569, pp. 158–186. Springer, Heidelberg (2009)
15. Vasconcelos, V.T.: Lambda and pi calculi, CAM and SECD machines. Journal of Functional Programming 15(1), 101–127 (2005)
16. Walker, D.: Substructural Type Systems. In: Advanced Topics in Types and Programming Languages. MIT Press, Cambridge (2005)
17. Yoshida, N., Vasconcelos, V.T.: Language primitives and type discipline for structured communication-based programming revisited. In: SecReT 2007. ENTCS, vol. 171(4), pp. 73–93. Elsevier Science Publishers, Amsterdam (2007)

Generic Forward and Backward Simulations II: Probabilistic Simulation

Ichiro Hasuo

[1] RIMS, Kyoto University, Japan
[2] PRESTO Research Promotion Program, Japan Science and Technology Agency

Abstract. Jonsson and Larsen's notion of probabilistic simulation is studied from a coalgebraic perspective. The notion is compared with two generic coalgebraic definitions of simulation: Hughes and Jacobs' one, and the one introduced previously by the author. We show that the first almost coincides with the second, and that the second is a special case of the last. We investigate implications of this characterization; notably the Jonsson-Larsen simulation is shown to be *sound*, i.e. its existence implies trace inclusion.

1 Introduction

Use of probabilistic algorithms in distributed and concurrent applications is common practice. Consequently, modeling and verification techniques for probabilistic systems have been extensively developed. One fundamental branch therein is about probabilistic *(bi)simulation*: it gives an answer when a probabilistic system is "equivalent" to another, or when one "refines" another.

In this paper we focus on simulation notions for purely probabilistic systems.[1] For such systems it is standard to define a notion of simulation using *weight functions*. The idea is first devised by Jonsson and Larsen [12]; it has inspired a large body of work including [1]. Our aim in this paper is to shed fresh, mathematical light on the idea, from the viewpoint of *coalgebra*.

Coalgebra is a mathematical/categorical presentation of state-based systems. Its initial success was brought about by a generic, coalgebraic characterization of bisimulation that applies to a variety of systems, including probabilistic ones (see e.g. [11, 17, 19]). The theory has since been extended to include various aspects of concurrency theory—such as SOS and modal logic (see e.g. [13]). Simulation, as "one-sided bisimulation," is one of such aspects.

Two approaches have been presented towards a coalgebraic theory of simulation: Hughes and Jacobs' [10] and the current author's [5]. Both approaches are generic, applicable to non-deterministic systems like LTS as well as probabilistic ones. In this paper we restrict them to a purely probabilistic setting and conduct a comparative study. The comparison is among the *Jonsson-Larsen simulation*, the *Hughes-Jacobs simulation*, and the one in the author's previous work [5] which we call the *Kleisli simulation*.

Among the three, the notion of Kleisli simulation is the most distinguishable: it is given not as a relation but as a function $X \to \mathcal{D}Y$, where X and Y are the state spaces

[1] Unlike e.g. Segala's probabilistic automata [18], they do not feature non-determinism.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 447–461, 2010.

of the involved systems. Therefore it is not suitable as a candidate of a *refinement relation*, the original motivation for the Jonsson-Larsen one. The Kleisli simulation rather follows the spirit of Lynch and Vaandrager [16]: it is a powerful tool for showing *trace inclusion*. While a direct proof of trace inclusion involves transitions within arbitrary many steps, finding a simulation is a stepwise matter. Indeed the notion of Kleisli simulation is precisely a coalgebraic generalization of the one in [16]; the former comes with the *forward* and *backward* variations just like the latter. The theory in [5] has been successfully applied to verification of probabilistic anonymity in [9].

Our findings are as follows. The standard Jonsson-Larsen simulation, defined in §3 concretely for a specific kind of probabilistic systems (we describe them in §2), is identified with a slightly restricted variant of the Hughes-Jacobs simulation (§4). This allows us to remove the unnecessary restriction that was hidden in the original concrete definition (§4.4), as well as provides a guideline in transferring the definition to other kinds of probabilistic systems (§4.5). On another link in the triangle, we identify the Hughes-Jacobs simulation as a special case of the Kleisli simulation (§5). From the generic *soundness theorem* [5]—existence of a Kleisli simulation implies (finite) trace inclusion—we thus conclude soundness of the Hughes-Jacobs notion, hence of the Jonsson-Larsen one.

Our expedition will be in a leisurely pace. In particular, no categorical or coalgebraic prerequisites are assumed; they are introduced on our way, on a call-by-need basis.

Due to space limitation, most proofs are deferred to an extended version [7]. It has also a series of example systems for further comparison of different simulation notions,

Notations. A square in a diagram which is not filled means that it *commutes*, that is, the equality symbol $=$ is implicit in it.

A probability (sub)distribution γ over a set X is often denoted like a table: $[\,x \mapsto \gamma(x)\,]_{x\in X}$. When an entry $x \in X$ is missing in the table, the probability 0 is assigned. Hence for example, when $x_0 \in X$ is a fixed element, $[\,x_0 \mapsto 1\,]$ means the distribution γ such that $\gamma(x_0) = 1$ and $\gamma(x) = 0$ for $x \neq x_0$.

2 Probabilistic System

We will be mainly interested in two kinds of purely probabilistic systems—GPAs and DTMCs—which we now define formally.

Definition 2.1 (Generative probabilistic automaton, GPA). Let Ac be a fixed nonempty alphabet; we refer to its element as an *action*. A *generative probabilistic automaton (GPA)* over Ac is a triple $\mathcal{X} = (X, x_0, c)$ where

- X is a nonempty set of *states*;
- $x_0 \in X$ is a chosen state which is called the *initial* one; and
- $c : X \to \mathcal{D}(\{\checkmark\} + \mathsf{Ac} \times X)$ is a *transition function*. Here $\{\checkmark\}$ is a singleton; $+$ denotes the disjoint union; and $\mathcal{D}$ is the *subdistribution* operation such that for a set Y

$$\mathcal{D}Y \;=\; \{\gamma : Y \to [0,1] \mid \sum_{y\in Y} \gamma(y) \leq 1\} \;\; . \tag{1}$$

  Such $d \in \mathcal{D}Y$ is called a *sub*-distribution since its values add up to not more than 1, instead of precisely 1.

The subdistribution $c(x)$ tells the probabilistic behavior of a state x. The value $c(x)(a, x')$ with $a \in \mathsf{Ac}$ and $x' \in X$ is the probability with which x makes the action a and moves to x'; that is, $c(x)(a, x') = \Pr[x \xrightarrow{a} x']$. We interpret the symbol ✓ as *successful termination*; thus with the probability $c(x)(\checkmark) = \Pr[x \to \checkmark]$ the state x is led to successful termination. The remaining probability $1 - c(x)(\checkmark) - \sum_{a,x'} c(x)(a, x')$—which may be more than 0 since $c(x)$ is a subdistribution—is understood as the probability with which x gets into *deadlock*.

GPAs are said to be *generative*, in contrast to *reactive* systems whose transition function is given as, say, $c : \mathsf{Ac} \times X \to \mathcal{D}(\{\checkmark\} + X)$. They differ in whether an action is chosen by the system or by the environment; see [4].

GPAs can be thought of as a probabilistic variant of labeled transition systems (LTSs). DTMCs, which we introduce shortly, are then probabilistic Kripke frames. The notion is standard, see e.g. [14,1]. The definitions in the literature vary in details; the following one is adapted to fit the current context.

Definition 2.2 (Discrete-Time Markov Chain, DTMC). Let AP be a fixed set of *atomic propositions*. A *discrete-time Markov chain (DTMC)* over AP is a quadruple $\mathcal{X} = (X, x_0, l, p)$ where

- X is a nonempty set of *states*, among which $x_0 \in X$ is an *initial* state;
- $l : X \to \mathcal{P}(\mathsf{AP})$ is a *labeling function* where $\mathcal{P}$ denotes the powerset. This assigns to a state $x \in X$ the set $l(x)$ of atomic propositions that hold at x;
- $p : X \to \mathcal{D}X$ is a *transition function*, where $\mathcal{D}$ is the operation in (1).

A DTMC has labels on its states, while a GPA has labels on its transitions.

3 Jonsson-Larsen Simulation

For DTMC and its variants, a standard definition of simulation [12] uses *weight functions*. Here we present the definition for DTMC taken from [1]. This family of simulation notions—based on weight functions—will be called *Jonsson-Larsen simulation*.

Definition 3.1 (JL-simulation for DTMC). Let $\mathcal{X} = (X, x_0, l, p)$ and $\mathcal{Y} = (Y, y_0, m, q)$ be DTMCs. A *Jonsson-Larsen simulation (JL-simulation)* from $\mathcal{X}$ to $\mathcal{Y}$ is a relation $R \subseteq X \times Y$ which satisfies the following.

1. The initial states are related, that is, $x_0 R y_0$.
2. Related states satisfy the same atomic propositions: xRy implies $l(x) = m(y)$.
3. For each $x \in X$ and $y \in Y$ such that xRy, there exists a *weight function*

$$\Delta_{x,y} \;:\; \big(\{\bot\} + X\big) \times \big(\{\bot\} + Y\big) \longrightarrow [0, 1] \qquad \text{such that}$$

   (a) $\Delta_{x,y}(u, v) > 0$ implies either
      - $u = \bot$, or
      - $u = x' \in X$, $v = y' \in Y$ and $x'Ry'$;

   (b) $\Delta_{x,y}(\bot, \bot) + \sum_{y' \in Y} \Delta_{x,y}(\bot, y') = 1 - \sum_{x' \in X} p(x)(x')$;

   (c) for each $x' \in X$: $\quad \Delta_{x,y}(x', \bot) + \sum_{y' \in Y} \Delta_{x,y}(x', y') = p(x)(x')$;

   (d) $\Delta_{x,y}(\bot, \bot) + \sum_{x' \in X} \Delta_{x,y}(x', \bot) = 1 - \sum_{y' \in Y} q(y)(y')$;

   (e) for each $y' \in Y$: $\quad \Delta_{x,y}(\bot, y') + \sum_{x' \in X} \Delta_{x,y}(x', y') = q(y)(y')$.

Although illustration of the previous definition is found e.g. in [1, Ex. 14], the definition hardly seems as "canonical" or "intuitive" as other notions such as (bi)simulation for ordinary LTS. For example, the only asymmetry between $\mathcal{X}$ and $\mathcal{Y}$ is found in Cond. 3.(a) where u, not v, is allowed to be $\bot$. One might wonder if it is possible to weaken this condition. Weakening $=$ into $\subseteq$ in Cond. 2 looks like another possibility. It is not clear either how to adapt this definition to GPA. Even less clear is whether the adapted notion satisfies soundness—existence of a simulation implies trace inclusion—which is a natural property to expect.

What we do in the rest of the paper is to put the above definition in a coalgebraic context. First it will be identified with a restriction of Hughes and Jacobs' simulation *(HJ-simulation)* [10]. From this we immediately obtain natural generalizations of the original definition, which are hinted above. The generic theory in [10] can be used to conduct some "sanity checks" for the generalized definitions. Adaptation to GPA comes for free, too. After that we will identify HJ-simulation with a certain subclass of *Kleisli* simulation from [5]. Soundness of JL-simulation for GPA is its corollary.

4 Hughes-Jacobs Simulation

4.1 Coalgebraic Modeling

In the Hughes-Jacobs theory of coalgebraic simulation, a system is modeled as a B-*coalgebra*, which is a function c of the type on the right. The set X (which is arbitrary) is the system's *state space*; the operation B specifies the kind of transitional behavior exhibited by the system; and the function c determines the system's dynamic behavior. We now elaborate on the operation B, which takes a set X and returns another set BX.

$$\begin{tikzcd} BX \\ X \arrow[u, "c"] \end{tikzcd}$$

Roughly speaking, it is the operation B which determines what kind of systems we are talking about. One choice of B makes a B-coalgebra an LTS; another choice of B is for a deterministic automaton (DA); and so on. Specifically,

B	$\mathcal{P}(\mathsf{Ac}\times\_\,)$	$2\times(\_\,)^{\mathsf{Ac}}$	$(\mathsf{Ac}_{\mathsf{out}}\times\_\,)^{\mathsf{Ac}_{\mathsf{in}}}$	$\mathcal{D}\big(\{\checkmark\}+\mathsf{Ac}\times\_\,\big)$	$\mathcal{P}(\mathsf{AP})\times\mathcal{D}(\_\,)$
B-coalg.	LTS	DA	Mealy mach.	GPA	DTMC

When B is $\mathcal{D}\big(\{\checkmark\}+\mathsf{Ac}\times\_\,\big)$, a B-coalgebra is a function $c : X \to \mathcal{D}\big(\{\checkmark\}+\mathsf{Ac}\times X\big)$; this is precisely a GPA (Def. 2.1) without an explicit initial state. For $B = \mathcal{P}(\mathsf{AP})\times\mathcal{D}(\_\,)$, a B-coalgebra is a function $c : X \to \mathcal{P}(\mathsf{AP})\times\mathcal{D}X$, which is identified with a DTMC (without an initial state) via the following bijective projection-tupling correspondence.

$$\frac{X \longrightarrow \mathcal{P}(\mathsf{AP})\times\mathcal{D}X}{X \longrightarrow \mathcal{P}(\mathsf{AP}) \qquad X \longrightarrow \mathcal{D}X} \qquad\qquad \begin{array}{c} c \\ \mapsdown\cong \\ (\pi_1\circ c,\,\pi_2\circ c) \end{array} \qquad\qquad \begin{array}{c} \langle l,p\rangle := \lambda x.\,(\,l(x),\,p(x)\,) \\ \mapsup\cong \\ (l,p) \end{array}$$

Here π_i denotes the i-th projection; $\langle l,p\rangle$ denotes the *tupling* of l and p.

To develop a "theory of systems" on top of this modeling, an operation B needs to be a *functor*. Leaving its detailed treatment to literature like [11], what it means is that the operation B not only applies to sets (i.e. $X \mapsto BX$) but also to functions. That is,

$$B \ : \quad (X \xrightarrow{f} Y) \quad \longmapsto \quad (BX \xrightarrow{Bf} BY) \ .$$

Note the domain and the codomain of the resulting function Bf.

The previous examples of B have natural action on functions. For example, given a function $f : X \to Y$,

$$\mathcal{P}(\mathsf{Ac} \times X) \overset{\mathcal{P}(\mathsf{Ac}\times f)}{\longrightarrow} \mathcal{P}(\mathsf{Ac} \times Y) \ , \quad u \longmapsto \bigl\{ (a, f(x)) \bigm| (a,x) \in u \bigr\} \ ;$$
$$\mathcal{D}\bigl(\{\checkmark\} + \mathsf{Ac} \times X\bigr) \overset{\mathcal{D}(\{\checkmark\}+\mathsf{Ac}\times f)}{\longrightarrow} \mathcal{D}\bigl(\{\checkmark\} + \mathsf{Ac} \times Y\bigr) \ ,$$
$$\gamma \longmapsto \bigl[\, \checkmark \mapsto \gamma(\checkmark) \ , \quad (a,y) \mapsto \textstyle\sum_{x \in f^{-1}(\{y\})} \gamma(a,x) \,\bigr] \ ;$$
$$\mathcal{P}(\mathsf{AP}) \times \mathcal{D}X \overset{\mathcal{P}(\mathsf{AP})\times\mathcal{D}f}{\longrightarrow} \mathcal{P}(\mathsf{AP}) \times \mathcal{D}Y \ , \ (u,\gamma) \longmapsto \bigl(u, \ \bigl[y \mapsto \textstyle\sum_{x \in f^{-1}(\{y\})} \gamma(x) \bigr] \bigr).$$

To be precise, such B is a functor of the type $\mathbf{Sets} \to \mathbf{Sets}$, from the category $\mathbf{Sets}$ of sets and functions to itself. We make a formal definition for the record.

Definition 4.1 (Functor, coalgebra). A *functor* $B : \mathbf{Sets} \to \mathbf{Sets}$ consists of its action on sets $X \longmapsto BX$ and on functions $(X \overset{f}{\to} Y) \longmapsto (BX \overset{Bf}{\to} BY)$, for each X and f. This is subject to the following conditions:

$$B(X \overset{\mathrm{id}_X}{\to} X) = (BX \overset{\mathrm{id}_{BX}}{\to} BX) \ ; \qquad B(X \overset{f}{\to} Y \overset{g}{\to} U) = (BX \overset{Bf}{\to} BY \overset{Bg}{\to} BU) \ .$$

A *B-coalgebra* is a pair $(X, c : X \to BX)$ of a set and a function; we shall simply denote it by $X \overset{c}{\to} BX$.

The *functoriality* of B is crucial in the following definition of coalgebraic bisimulation (notice use of $B\pi_i$). The definition subsumes many known notions of bisimulation.

Definition 4.2. Let $B : \mathbf{Sets} \to \mathbf{Sets}$ be a functor and $c : X \to BX$ and $d : Y \to BY$ be B-coalgebras. A *coalgebraic bisimulation* is a relation $R \subseteq X \times Y$ such that: there exists a function $r : R \to BR$ that makes the diagram on the right commute. Here π_1 and π_2 are obvious projections.

$$\begin{tikzcd} BX & BR \arrow[l, "B\pi_1"'] \arrow[r, "B\pi_2"] & BY \\ X \arrow[u, "c"] & R \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] \arrow[u, "r"] & Y \arrow[u, "d"'] \end{tikzcd} \tag{2}$$

When B represents purely probabilistic systems such as DTMCs, the above coalgebraic bisimulation instantiates to the one that uses a weight function. It coincides with the more common formulation via equivalence classes [15]. The coincidence proof is implicit in [12, Thm. 4.6] and is much more systematically conducted in [19].

4.2 Hughes-Jacobs Simulation

Roughly speaking, simulation is "one-sided" bisimulation. When R is a simulation and xRy, we require y to exhibit "at least as much" behavior as x does, that is,

$$(x\text{'s behavior}) \ \sqsubseteq \ (y\text{'s behavior})$$

in terms of a suitable preorder $\sqsubseteq$ of "behavior inclusion." Hughes and Jacobs [10] used this intuition and defined generic simulation as a variant of Def. 4.2. In order to do so, a functor $B : \mathbf{Sets} \to \mathbf{Sets}$ needs to come with a "behavior-inclusion" preorder $\sqsubseteq$.

Definition 4.3 (Functor with preorder). A *functor with preorder* consists of a functor $B : \mathbf{Sets} \to \mathbf{Sets}$ and a class of preorders $\{\sqsubseteq_{BX}\}_X$ for each set X, where $\sqsubseteq_{BX}$ is on the set BX. Further, given a function $f : X \to Y$, its action $Bf : BX \to BY$ is required to be a monotone function. We often suppress the subscript in $\sqsubseteq_{BX}$.

Example 4.4. Let $B = \mathcal{P}(\mathsf{Ac} \times \_)$, for which B-coalgebras are LTSs. It is a functor with preorder, with a natural choice of $\sqsubseteq_{BX}$ being the inclusion order.

Let $B = \mathcal{D}\big(\{\checkmark\} + \mathsf{Ac} \times \_\big)$; a B-coalgebra is a GPA. For $\gamma, \delta \in BX$ we define

$$\gamma \sqsubseteq_{BX} \delta \quad \overset{\text{def.}}{\Longleftrightarrow} \quad \gamma(\checkmark) \le \delta(\checkmark) \quad \text{and} \quad \gamma(a,x) \le \delta(a,x) \quad \text{for each } a \text{ and } x.$$

Note that $\sqsubseteq_{BX}$ need not be reduced to the equality, since γ and δ are *sub*distributions.

Let $B = \mathcal{P}(\mathsf{AP}) \times \mathcal{D}(\_)$; a B-coalgebra is then a DTMC. There are a few natural candidates for the preorder $\sqsubseteq_{BX}$. One is:

$$(u,\gamma) \sqsubseteq^{=}_{BX} (v,\delta) \quad \overset{\text{def.}}{\Longleftrightarrow} \quad u = v \quad \text{and} \quad \gamma(x) \le \delta(x) \quad \text{for each } x.$$

We denote this order by $\sqsubseteq^{=}$. Noting that u and v are subsets of AP, we could replace the condition $u = v$ by $u \subseteq v$, for example. The resulting order will be denoted by $\sqsubseteq^{\subseteq}$.

Definition 4.5 (HJ-simulation). Let $(B, \sqsubseteq)$ be a functor with preorder, and $X \overset{c}{\to} BX, Y \overset{d}{\to} BY$ be B-coalgebras. A *Hughes-Jacobs simulation (HJ-simulation)* from c to d is a relation $R \subseteq X \times Y$ such that: there exists a function $r : R \to BR$ which makes the inequalities on the right hold.

$$\begin{tikzcd}
BX & BR \arrow[l, "B\pi_1"'] \arrow[r, "B\pi_2"] & BY \\
X \arrow[u, "c"] \arrow[ur, phantom, "\sqsubseteq"] & R \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] \arrow[u, "r"] \arrow[ur, phantom, "\sqsubseteq"] & Y \arrow[u, "d"']
\end{tikzcd} \tag{3}$$

That is to be precise: for each $(x,y) \in R$

$$(c \circ \pi_1)(x,y) \sqsubseteq_{BX} (B\pi_1 \circ r)(x,y) \quad \text{and} \quad (B\pi_2 \circ r)(x,y) \sqsubseteq_{BY} (d \circ \pi_2)(x,y) \ .$$

The formulation is different from the original one [10] where a lax relation lifting is used. The equivalence is proved in [7, Appendix A.1].

4.3 Jonsson-Larsen Simulation as Hughes-Jacobs Simulation

Here is the first main observation in this paper: JL is HJ. The order $\sqsubseteq^{=}$ is from Ex. 4.4.

Theorem 4.6. *Let $\mathcal{X} = (X, x_0, l, p)$ and $\mathcal{Y} = (Y, y_0, m, q)$ be DTMCs, and let R be a JL-simulation from $\mathcal{X}$ to $\mathcal{Y}$. Then R is a HJ-simulation from the coalgebra $\langle l, p\rangle$ to $\langle m, q\rangle$: there exists r that makes the following (in)equalities hold.*

$$\begin{tikzcd}
\mathcal{P}(\mathsf{AP}) \times \mathcal{D}X & \mathcal{P}(\mathsf{AP}) \times \mathcal{D}R \arrow[l, "\mathcal{P}(\mathsf{AP}) \times \mathcal{D}\pi_1"'] \arrow[r, "\mathcal{P}(\mathsf{AP}) \times \mathcal{D}\pi_2"] & \mathcal{P}(\mathsf{AP}) \times \mathcal{D}Y \\
X \arrow[u, "\langle l,p\rangle"] \arrow[ur, phantom, "="] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[u, "r"] \arrow[ur, phantom, "\sqsubseteq^{="] & Y \arrow[u, "\langle m,q\rangle"']
\end{tikzcd}$$

□

We include initial states (Cond. 1, Def. 3.1) and obtain the following characterization.

Theorem 4.7 (JL is HJ). *Let $\mathcal{X}$ and $\mathcal{Y}$ be DTMCs in Thm. 4.6. A relation $R \subseteq X \times Y$ is a JL-simulation if and only if there exist functions r and r_0 that make the following (in)equalities hold. The set $\{*\}$ is a singleton.*

$$\begin{tikzcd}
\mathcal{P}(\mathsf{AP}) \times \mathcal{D}X \arrow[dr, phantom, "="] & \mathcal{P}(\mathsf{AP}) \times \mathcal{D}R \arrow[l, "\mathcal{P}(\mathsf{AP})\times\mathcal{D}\pi_1"'] \arrow[r, "\mathcal{P}(\mathsf{AP})\times\mathcal{D}\pi_2"] \arrow[dr, phantom, "\sqsubseteq^{=}"] & \mathcal{P}(\mathsf{AP}) \times \mathcal{D}Y \\
X \arrow[u, "\langle l,p\rangle"] \arrow[dr, phantom, "="] & R \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] \arrow[u, "r"] \arrow[dr, phantom, "="] & Y \arrow[u, "\langle m,q\rangle"'] \\
 & \{*\} \arrow[u, "r_0"] \arrow[ul, bend left, "x_0"] \arrow[ur, bend right, "y_0"'] &
\end{tikzcd} \tag{4}$$

Note that a function $x_0 : \{*\} \to X$ can be identified with an element $x_0 \in X$. □

4.4 Generalized Jonsson-Larsen Simulation

Thm. 4.7 shows that JL-simulation does not reach the full generality of HJ-simulation: the top-left square is an equality in (4), which is not necessary. Translating HJ-simulation into the Jonsson-Larsen style concrete terms, we are led to the following definition.

Definition 4.8 (JL'-simulation for DTMC). A *JL'-simulation* is the same thing as a JL-simulation (Def. 3.1) except for the following.

- A weight function is of the type $\Delta_{x,y} \ : \ \big(\{\bot\} + X\big) \times \big(\{\bot\} + Y\big) \longrightarrow [-1, 1]$.
- Cond. 3.(a) is weakened: the value of $\Delta_{x,y}(u, v)$ must lie in the following range, according to u and v:
  - when $u = x' \in X$ and $v = y' \in Y$, if $x'Ry'$ then $\Delta_{x,y}(x', y') \geq 0$; if $(x', y') \notin R$ then $\Delta_{x,y}(x', y') = 0$;
  - $\Delta_{x,y}(\bot, y') \geq 0$ for each $y' \in Y$;
  - $\Delta_{x,y}(x', \bot) \leq 0$ for each $x' \in X$;
  - $\Delta_{x,y}(\bot, \bot)$ can be positive, zero or negative.
- Cond. 3.(b) and 3.(d) are dropped.

$$\begin{array}{c||c|c}
{}_{u}\backslash^{v} & \bot & \cdots\ y'\ \cdots \\
\hline\hline
\bot & \lesseqgtr 0 & \geq 0 \\
\hline
\vdots & & \\
x' & \leq 0 & \begin{cases} \geq 0 & (x'Ry') \\ 0 & (\text{o.w.}) \end{cases} \\
\vdots & &
\end{array}$$

Now a weight function can take negative values. Cond. 3.(b) and 3.(d) played no role in Thm. 4.6, hence are dropped. Similarly to JL-simulation, finding a weight function is filling in the matrix above on the right, in such a way that its rows and columns add up to the right values like $p(x)(x')$ or $q(y)(y')$. The task is easier with JL'-simulation because each entry can be picked from a broadened domain.

One can further generalize the previous definition by replacing $\sqsubseteq^{=}$ by $\sqsubseteq^{\subseteq}$ (Ex. 4.4): in this case the system $\mathcal{X}$ to be simulated satisfies no more atomic propositions than $\mathcal{Y}$ does. This generalization is useful e.g. when we are interested in safety properties, and atomic propositions represent systems' actions.

Definition 4.9 (JL"-simulation for DTMC). A *JL"-simulation* is the same as a JL'-simulation (Def. 4.8), except that Cond. 2 is replaced by

2. xRy implies $l(x) \subseteq m(y)$.

Proposition 4.10. *Let $\mathcal{X}$ and $\mathcal{Y}$ be DTMCs as in Thm. 4.6, and $R \subseteq X \times Y$.*

1. *The relation R is a JL'-simulation if and only if there exist r and r_0 that validate the (in)equalities in (4), with the top-left equality replaced by $\sqsubseteq^{=}$.*
2. *The relation R is a JL"-simulation if and only if there exist r and r_0 that validate the diagram (4), with the top two squares filled with $\sqsubseteq^{\subseteq}$.* □

Let us do some sanity checks. The following holds for JL" instead of JL' too; also for the conventional notion of JL simulation (see [1]).

Proposition 4.11. *Let $\mathcal{X} = (X, x_0, l, p)$ and $\mathcal{Y} = (Y, y_0, m, q)$ be DTMCs.*

1. *If $R \subseteq X \times Y$ is a bisimulation, then R and R^{op} are both JL'-simulations.*
2. *The family of JL'-simulations from $\mathcal{X}$ to $\mathcal{Y}$ is closed under arbitrary unions. Therefore there is the largest JL'-simulation $\lesssim_{JL'}$, called* JL'-similarity.
3. *JL'-simulations are closed under composition. Hence $\lesssim_{JL'}$ is transitive.*

Proof. We apply Lem. 4.2 and Prop. 5.4 of [10]. This involves checking a technical condition of *stability* of orders. See [7]. □

4.5 Jonsson-Larsen Simulation for GPA

Another implication of Thm. 4.7 is adaptation of JL-simulation for other kinds of probabilistic systems, via HJ-simulation which is general by definition.

Definition 4.12 (JL-simulation for GPA). Let $\mathcal{X} = (X, x_0, c)$ and $\mathcal{Y} = (Y, y_0, d)$ be GPAs. A *JL-simulation* from $\mathcal{X}$ to $\mathcal{Y}$ is a relation $R \subseteq X \times Y$ such that:

1. The initial states are related, that is, $x_0 R y_0$.
2. For each pair $(x, y) \in R$, there exists a weight function

$$\Delta_{x,y} \;:\; \bigl(\{\bot\}+\{\checkmark\}+\mathsf{Ac}\times X\bigr)\times\bigl(\{\bot\}+\{\checkmark\}+\mathsf{Ac}\times Y\bigr) \longrightarrow [-1,1] \quad \text{such that}$$

(a) $\Delta_{x,y}(u, v)$ lies in the range on the right. In particular, $\Delta_{x,y}((a, x'), (a', y')) > 0$ only if $a = a'$ and $x' R y'$;

$_u\backslash^v$	$\bot$	$\checkmark$	$\cdots\ (a_1, y'_1)\ \cdots$	$\cdots\ (a_2, y'_2)\ \cdots$
$\bot$	$\lesseqgtr 0$	≥ 0	≥ 0	≥ 0
$\checkmark$	≤ 0	≥ 0	0	0
$\vdots$ (a_1, x'_1) $\vdots$	≤ 0	0	$\begin{cases} \geq 0 & (x'_1 R y'_1) \\ 0 & (\text{o.w.}) \end{cases}$	0
$\vdots$ (a_2, x'_2) $\vdots$	≤ 0	0	0	$\begin{cases} \geq 0 & (x'_2 R y'_2) \\ 0 & (\text{o.w.}) \end{cases}$

(b) $c(x)(\checkmark) = \Delta_{x,y}(\checkmark, \bot) + \Delta_{x,y}(\checkmark, \checkmark)$;
(c) $c(x)(a, x') = \Delta_{x,y}((a, x'), \bot) + \sum_{y'} \Delta_{x,y}((a, x'), (a, y'))$ for each a and x';
(d) $d(x)(\checkmark) = \Delta_{x,y}(\bot, \checkmark) + \Delta_{x,y}(\checkmark, \checkmark)$;
(e) $d(y)(a, y') = \Delta_{x,y}(\bot, (a, y')) + \sum_{x'} \Delta_{x,y}((a, x'), (a, y'))$ for each a and y'.

This definition seems to appear for the first time. It coincides with HJ-simulation for $B = \mathcal{D}(\{\checkmark\} + \mathsf{Ac} \times (\_))$, with B equipped with the order in Ex. 4.4 (the proof is easy). Properties like in Prop. 4.11 hold as well. Remaining is the issue of *soundness*; it is not obvious at all from the above complicated definition. One of our main contributions is the soundness proof later in §5.7, which uses the generic theory in [5].

5 Kleisli Forward and Backward Simulation

We now describe the third kind of simulation from [5]. We shall refer to this family as *Kleisli simulation*, for the reason that is explained shortly. Kleisli simulation consists of four subclasses: *forward*, *backward*, and two *hybrid* ones, like in [16]. The most notable difference from JL- and HJ-notions is that a Kleisli simulation is itself not a relation.

5.1 Kleisli Arrow

First we fix our domain of discourse—*Kleisli arrows*. They are arrows in a Kleisli category, a standard categorical construct. Our description is however in concrete terms.

Definition 5.1 (Kleisli arrow). Let X and Y be arbitrary sets. A *Kleisli arrow* from X to Y, denoted by $f : X \nrightarrow Y$, is a function $f : X \to \mathcal{D}Y$. A few typical Kleisli arrows:

- The Kleisli arrow $\eta_X : X \nrightarrow X$, for each X, is the function $\eta_X : X \to \mathcal{D}X$ that carries $x \in X$ to $[x \mapsto 1]$.
- Given consecutive Kleisli arrows $X \overset{f}{\nrightarrow} Y$ and $Y \overset{g}{\nrightarrow} U$, we have $g \odot f : X \nrightarrow U$ by
$$g \odot f \ : \ X \longrightarrow \mathcal{D}U \quad , \quad x \longmapsto \lambda u.\ \textstyle\sum_{y \in Y} g(y)(u) \cdot f(x)(y) \ .$$
- For each (ordinary) function $f : X \to Y$, we have $Jf : X \nrightarrow Y$ defined by $X \xrightarrow{f} Y \xrightarrow{\eta_Y} \mathcal{D}Y$. That is, $(Jf)(x) = [f(x) \mapsto 1]$. This generalizes η_X by: $\eta_X = J(\mathrm{id}_X)$.

The following are straightforward; they say that Kleisli arrows form a category.

Proposition 5.2. *1. Composition of Kleisli arrows is associative: for three consecutive Kleisli arrows $X \overset{f}{\nrightarrow} Y \overset{g}{\nrightarrow} U \overset{h}{\nrightarrow} V$, we have $h \odot (g \odot f) = (h \odot g) \odot f$.*

2. η is the unit of composition: for $X \overset{f}{\nrightarrow} Y$ we have $\eta_Y \odot f = f = f \odot \eta_X$. □

One can think of a Kleisli arrow $f : X \nrightarrow Y$ as a "function from X to Y, with implicit probabilistic branching"; or as a "probabilistic computation of input type X and output type Y." The operator $\odot$ realizes natural composition of such probabilistic computations. The embedding Jf of an ordinary function endows f with trivial branching.

There is a natural order between parallel Kleisli arrows.

Definition 5.3. Between a parallel pair of Kleisli arrows $f, g : X \nrightarrow Y$, we define an order $f \sqsubseteq g$ if: $f(x)(y) \le g(x)(y)$ for each $x \in X$ and $y \in Y$.

5.2 Probabilistic Systems as Kleisli Coalgebras

A GPA $\mathcal{X} = (X, x_0, c)$ (Def. 2.1) can be presented by two Kleisli arrows:

$$\{*\} \overset{Jx_0}{\longrightarrow\!\!\!\!\!\!\!+\;\;} X \overset{c}{\longrightarrow\!\!\!\!\!\!\!+\;\;} \{\checkmark\} + \mathsf{Ac} \times X \ . \tag{5}$$

This is a prototype of the kind of systems on which we define Kleisli simulation. First we parametrize the '$\{\checkmark\} + \mathsf{Ac} \times (\_)$' part in the above.

Definition 5.4 (Polynomial functor). A *polynomial functor* is a functor $F : \mathbf{Sets} \to \mathbf{Sets}$ which is constructed

- from the identity functor $(\_)$ and the constant functor C for each set C,
- using finite products and arbitrary disjoint union (i.e. coproduct).

In the BNF notation: $F ::= (\_) \mid C \mid F_1 \times F_2 \mid \coprod_{i \in I} F_i$.

The functor $\{\checkmark\} + \mathsf{Ac} \times (\_)$ is polynomial; so is e.g. $(\mathsf{Ac} + \_)^* = \coprod_{n<\omega} (\mathsf{Ac} + \_)^n$.

Lemma 5.5. *A polynomial functor F has canonical action on Kleisli arrows, carrying $X \overset{f}{\nrightarrow} Y$ to $FX \overset{Ff}{\nrightarrow} FY$.*

Proof. A general categorical proof is found in [8, §2.2]; one can also define such action concretely by induction on the construction of F. □

In most cases F's action on Kleisli arrows is obvious. For $F = \{\checkmark\} + \mathsf{Ac} \times (\_)$ and $f : X \nrightarrow Y$, the Kleisli arrow $Ff : FX \nrightarrow FY$ is given by the function $\{\checkmark\} + \mathsf{Ac} \times X \longrightarrow \mathcal{D}(\{\checkmark\} + \mathsf{Ac} \times Y)$, defined by

$$\checkmark \longmapsto [\, \checkmark \mapsto 1 \,] \ , \quad (a, x) \longmapsto [\, (a, y) \mapsto f(x)(y) \,]_{y \in Y} \ .$$

Definition 5.6 (Probabilistic F-system). Let F be a polynomial functor. A *probabilistic F-system* (or simply *F-system*) is a triple $\mathcal{X} = (X, s, c)$, where X is an arbitrary set and $\{*\} \overset{s}{\longrightarrow\!\!\!\!\!\!\!\!\!+\,\,\,\,} X \overset{c}{\longrightarrow\!\!\!\!\!\!\!\!\!+\,\,\,\,} FX$ are two Kleisli arrows. Recall that probabilistic branching is implicit in Kleisli arrows.

Example 5.7. A GPA induces an F-system, with $F = \{\checkmark\} + \mathsf{Ac} \times (\_)$; see (5). F-system is more general than GPA since the former allows a subdistribution on initial states (i.e. $s \in \mathcal{D}X$) rather than a single initial state. This additional generality is however not important.

A DTMC cannot be seen as an F-system as it is: its dynamics is given by a function $X \overset{\langle l,p \rangle}{\to} \mathcal{P}(\mathsf{AP}) \times \mathcal{D}X$ which cannot be understood as a Kleisli arrow. We can fix it by moving "state labels" into "transition labels." Let us define a function $c_{l,p}$ by

$$c_{l,p} \ : \ X \longrightarrow \mathcal{D}\big(\, \mathcal{P}(\mathsf{AP}) \times X \,\big) \ , \quad x \longmapsto \Big[\, (l(x), x') \ \mapsto \ p(x)(x') \,\Big]_{x' \in X} \ ;$$

then the F-system $\{*\} \overset{Jx_0}{\nrightarrow} X \overset{c_{l,p}}{\nrightarrow} \mathcal{P}(\mathsf{AP}) \times X$ represents a DTMC (X, x_0, l, p).

The notion of (probabilistic) F-system is essentially a *Kleisli F-coalgebra* $X \overset{c}{\nrightarrow} FX$ equipped with an explicit initial state $\{*\} \overset{s}{\nrightarrow} X$. In coalgebraic studies it is usually unnecessary to speak about explicit initial states; we however need that in this paper for formulating the soundness result (Thm. 5.20). See [6, §3.2.4].

Let us compare the current *Kleisli coalgebraic modeling* of GPAs (Ex. 5.7) with the modeling in §4.1. They are the same in that the dynamics of a GPA is represented by a function $X \to \mathcal{D}(\{\checkmark\} + \mathsf{Ac} \times X)$. In the Kleisli modeling, the functor $B = \mathcal{D}(\{\checkmark\} + \mathsf{Ac} \times (\_))$ is divided into $\mathcal{D}$ (*branching* part) and $F = \{\checkmark\} + \mathsf{Ac} \times (\_)$ (*transition/action* part); the former is then "thrown under the rug" using Kleisli arrows.

5.3 Kleisli Simulation

Definition 5.8 (Kleisli simulation). Let F be a polynomial functor and $\mathcal{X} = (X, s, c)$ and $\mathcal{Y} = (Y, t, d)$ be F-systems. A *forward Kleisli simulation* from $\mathcal{X}$ to $\mathcal{Y}$ is a Kleisli arrow $f : Y \nrightarrow X$ such that $c \odot f \sqsubseteq (Ff) \odot d$ and $s \sqsubseteq f \odot t$ (see below left). Note the direction of f. It is also called simply a *forward simulation*.

fwd.
$$\begin{tikzcd}
FX & & FY \arrow[ll, "Ff"', "\shortmid" marking] \\
X \arrow[u, "c", "\shortmid" marking] \arrow[urr, phantom, "\sqsubseteq"] & & Y \arrow[ll, "f"', "\shortmid" marking] \arrow[u, "d"', "\shortmid" marking] \\
& \{*\} \arrow[ul, bend left, "s", "\shortmid" marking] \arrow[ur, bend right, "t"', "\shortmid" marking] \arrow[u, phantom, "\sqsubseteq"] &
\end{tikzcd}$$

bwd.
$$\begin{tikzcd}
FX \arrow[rr, "Fb", "\shortmid" marking] & & FY \\
X \arrow[u, "c", "\shortmid" marking] \arrow[rr, "b", "\shortmid" marking] \arrow[urr, phantom, "\sqsubseteq"] & & Y \arrow[u, "d"', "\shortmid" marking] \\
& \{*\} \arrow[ul, bend left, "s", "\shortmid" marking] \arrow[ur, bend right, "t"', "\shortmid" marking] \arrow[u, phantom, "\sqsubseteq"] &
\end{tikzcd}$$

A *backward (Kleisli) simulation* is a Kleisli arrow $b : X \nrightarrow Y$ such that $(Fb) \odot c \sqsubseteq d \odot b$ and $b \odot s \sqsubseteq t$ (see above right). Here the order $\sqsubseteq$ refers to the one in Def. 5.3.

In fact, the last definition is an instance of *generic forward and backward simulation* in [5,6]. The general definition has an extra parameter T that specifies a *branching type*. It is fixed to $T = \mathcal{D}$ in this paper, representing probabilistic branching. Another main example is $T = \mathcal{P}$, the powerset operation, for *non-deterministic* branching.

This extra parameter T is used in the definition of Kleisli arrow. Namely, $f : X \nrightarrow Y$ is defined to be a function $f : X \to TY$. When $T = \mathcal{P}$, a Kleisli arrow $f : X \nrightarrow Y$ can be identified with a *binary relation* $R_f \subseteq X \times Y$: xR_fy if and only if $y \in f(x)$. In this case, if moreover $F = \mathsf{Ac} \times (\_)$ for which F-systems are ordinary LTSs, Kleisli simulation (Def. 5.8) coincides with the standard notions of forward and backward simulation for LTS (see e.g. [16]). To summarize: probabilistic Kleisli simulation (Def. 5.8) is a natural generalization of non-deterministic simulation in [16].

5.4 Kleisli Simulation for GPA

We further instantiate the definition to GPA, i.e. $F = \{\checkmark\} + \mathsf{Ac} \times (\_)$. It demonstrates Kleisli simulation's affinity to the conventional simulation notions for LTS.

Notation 5.9. A forward simulation is a function $f : Y \to \mathcal{D}X$; we write $\Pr[y \dashrightarrow x]$ for the value $f(y)(x)$. We let $\Pr[x \to \checkmark]$ and $\Pr[x \xrightarrow{a} x']$ have their obvious meanings. We also compose events; for example

$$\Pr[y \dashrightarrow x \xrightarrow{a} x'] \;:=\; \Pr[y \dashrightarrow x] \cdot \Pr[x \xrightarrow{a} x'] \;=\; f(y)(x) \cdot c(x)(a, x') \enspace .$$

For a backward simulation, we write $\Pr[x \dashrightarrow y]$ for $b(x)(y)$.

Definition 5.10 (Forward simulation for GPA). Let $\mathcal{X} = (X, x_0, c)$ and $\mathcal{Y} = (Y, y_0, d)$ be GPAs. A *forward (Kleisli) simulation* from $\mathcal{X}$ to $\mathcal{Y}$ is a function $f : Y \to \mathcal{D}X$ which satisfies the following (in)equalities.

$$\Pr[y_0 \dashrightarrow x_0] = 1 \qquad \text{(INIT)}$$

$$\textstyle\sum_{x \in X} \Pr[y \dashrightarrow x \to \checkmark] \le \Pr[y \to \checkmark] \qquad \text{for each } y \in Y \qquad \text{(TERM)}$$

$$\textstyle\sum_{x \in X} \Pr[y \dashrightarrow x \xrightarrow{a} x'] \le \sum_{y' \in Y} \Pr[y \xrightarrow{a} y' \dashrightarrow x']$$
$$\text{for each } y \in Y,\ a \in \mathsf{Ac} \text{ and } x' \in X \qquad \text{(ACT)}$$

The condition (ACT) is illuminating. It can be depicted as the below left, which bears a clear affinity to the standard non-deterministic condition shown on the right.

$$\mathrm{Pr}\left[\begin{tikzcd} y \arrow[d, dotted] & \\ \bullet \arrow[r, "a"] & x' \end{tikzcd}\right] \le \mathrm{Pr}\left[\begin{tikzcd} y \arrow[r, "a"] & \bullet \arrow[d, dotted] \\ & x' \end{tikzcd}\right] \qquad \left(\begin{tikzcd} y \arrow[d, dotted, dash] & \\ \bullet \arrow[r, "a"] & x' \end{tikzcd}\right) \text{ implies } \left(\begin{tikzcd} y \arrow[r, "a"] & \exists\bullet \arrow[d, dotted, dash] \\ & x' \end{tikzcd}\right)$$

Definition 5.11 (Backward simulation for GPA). A *backward (Kleisli) simulation* from $\mathcal{X}$ to $\mathcal{Y}$ is a function $b : X \to \mathcal{D}Y$ which satisfies the following inequalities.

$$\mathrm{Pr}[x_0 \dashrightarrow y_0] \le 1 \qquad (\textsc{Init})$$

$$\mathrm{Pr}[x \to \checkmark] \le \textstyle\sum_{y \in X} \mathrm{Pr}[x \dashrightarrow y \to \checkmark] \quad \text{for each } x \in X \qquad (\textsc{Term})$$

$$\textstyle\sum_{x' \in X} \mathrm{Pr}[x \xrightarrow{a} x' \dashrightarrow y'] \le \textstyle\sum_{y \in Y} \mathrm{Pr}[x \dashrightarrow y \xrightarrow{a} y'] \quad \text{for each } x \in X,\, a \in \mathsf{Ac} \text{ and } y' \in Y \qquad (\textsc{Act})$$

5.5 Hughes-Jacobs Simulation as Hybrid Kleisli Simulation

Definition 5.12 (Hybrid simulation). Let $\mathcal{X} = (X, s, c)$ and $\mathcal{Y} = (Y, t, d)$ be F-systems. A *forward-backward (Kleisli) simulation* is a triple $(\mathcal{U}, f, b)$ where

- $\mathcal{U} = (U, u, e)$ is an F-system called the *intermediate system*;
- f is a forward simulation from $\mathcal{X}$ to $\mathcal{U}$, and
- b is a backward simulation from $\mathcal{U}$ to $\mathcal{Y}$. See below on the left.

<u>fwd.-bwd.</u>

$$\begin{tikzcd} FX & FU \arrow[l, "Ff"', "\shortmid" marking] \arrow[r, "Fb", "\shortmid" marking] & FY \\ X \arrow[u, "c", "\shortmid" marking] \arrow[ur, phantom, "\sqsubseteq"] & U \arrow[l, "f"', "\shortmid" marking] \arrow[r, "b", "\shortmid" marking] \arrow[u, "e"', "\shortmid" marking] \arrow[ur, phantom, "\sqsubseteq"] & Y \arrow[u, "d"', "\shortmid" marking] \\ & \{*\} \arrow[ul, bend left, "s", "\shortmid" marking] \arrow[u, "u"', "\shortmid" marking] \arrow[ur, bend right, "t"', "\shortmid" marking] \arrow[ul, phantom, "\sqsubseteq"] \arrow[ur, phantom, "\sqsubseteq"] & \end{tikzcd}$$

<u>bwd.-fwd.</u>

$$\begin{tikzcd} FX \arrow[r, "Fb", "\shortmid" marking] & FU & FY \arrow[l, "Ff"', "\shortmid" marking] \\ X \arrow[u, "c", "\shortmid" marking] \arrow[r, "b", "\shortmid" marking] \arrow[ur, phantom, "\sqsubseteq"] & U \arrow[u, "e"', "\shortmid" marking] \arrow[ur, phantom, "\sqsubseteq"] & Y \arrow[l, "f"', "\shortmid" marking] \arrow[u, "d"', "\shortmid" marking] \\ & \{*\} \arrow[ul, bend left, "s", "\shortmid" marking] \arrow[u, "u"', "\shortmid" marking] \arrow[ur, bend right, "t"', "\shortmid" marking] \arrow[ul, phantom, "\sqsubseteq"] \arrow[ur, phantom, "\sqsubseteq"] & \end{tikzcd}$$

Similarly, a *backward-forward (Kleisli) simulation* is a triple $(\mathcal{U}, b, f)$ of an intermediate system $\mathcal{U}$, a backward simulation b from $\mathcal{X}$ to $\mathcal{U}$, and a forward simulation f from $\mathcal{U}$ to $\mathcal{Y}$. See above on the right.

Proposition 5.13. *Let $\mathcal{X}, \mathcal{Y}$ be F-systems. If there is a non-hybrid simulation from $\mathcal{X}$ to $\mathcal{Y}$, then there are both fwd.-bwd. and bwd.-fwd. ones from $\mathcal{X}$ to $\mathcal{Y}$.*

Proof. A forward simulation f from $\mathcal{X}$ to $\mathcal{Y}$ induces a backward-forward simulation $(\mathcal{X}, J(\mathrm{id}), f)$; it has $\mathcal{X}$ itself as an intermediate system. The other cases are similar. □

Lemma 5.14. *Let F be a polynomial functor. Then the functor $\mathcal{D}F$ has the following natural order. This makes $(\mathcal{D}F, \sqsubseteq_{\mathcal{D}F})$ a functor with preorder (Def. 4.3).*

$$\gamma \sqsubseteq_{\mathcal{D}FX} \delta \quad \overset{\text{def.}}{\Longleftrightarrow} \quad \gamma(u) \le \delta(u) \quad \textit{for all } u \in FX. \qquad \square$$

When $F = \{\checkmark\} + \mathsf{Ac} \times (\_)$ and $B = \mathcal{D}F$, both B-coalgebras and probabilistic F-systems represent GPAs. In this case, the order on B in the previous definition coincides with the one in Ex. 4.4.

Here comes our second main observation.

Theorem 5.15 (HJ is Kleisli). *Let $X \xrightarrow{c} \mathcal{D}FX$ and $Y \xrightarrow{d} \mathcal{D}FY$ be $\mathcal{D}F$-coalgebras, $x_0 \in X$ and $y_0 \in Y$ be chosen (initial) states, and $R \subseteq X \times Y$ be a relation. Assume that there exists a function r that validates the inequalities in the diagram on the left, that is, that R is a HJ-simulation from c to d such that $x_0 R y_0$.*

$$\begin{tikzcd}
\mathcal{D}FX & \mathcal{D}FR \arrow[l, "\mathcal{D}F\pi_1"'] \arrow[r, "\mathcal{D}F\pi_2"] & \mathcal{D}FY & & FX & FR \arrow[l, "F(J\pi_1)"'] \arrow[r, "F(J\pi_2)"] & FY \\
X \arrow[u, "c"] \arrow[ur, phantom, "\sqsubseteq_{\mathcal{D}F}"] & R \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] \arrow[u, "r"] \arrow[ur, phantom, "\sqsubseteq_{\mathcal{D}F}"] & Y \arrow[u, "d"'] & \Longrightarrow & X \arrow[u, "c"] \arrow[ur, phantom, "\sqsubseteq"] & R \arrow[l, "J\pi_1"] \arrow[r, "J\pi_2"'] \arrow[u, "r"] \arrow[ur, phantom, "\sqsubseteq"] & Y \arrow[u, "d"'] \\
& \{*\} \arrow[ul, bend left, "x_0"] \arrow[u, "\langle x_0,y_0\rangle"] \arrow[ur, bend right, "y_0"'] & & & & \{*\} \arrow[ul, bend left, "Jx_0"] \arrow[u, "J\langle x_0,y_0\rangle"] \arrow[ur, bend right, "Jy_0"'] &
\end{tikzcd} \tag{6}$$

Then we have a fwd.-bwd. simulation from the F-system (X, Jx_0, c) to (Y, Jy_0, d), shown above on the right. Note the order $\sqsubseteq$ therein refers to the one in Def. 5.3. □

In short: a HJ-simulation between $\mathcal{D}F$-coalgebras induces a fwd.-bwd. simulation between the corresponding F-systems. The proof is found in [7].

Fwd.-bwd. simulation instantiates to GPA, like in §5.4. Thm. 4.7 yields:

Corollary 5.16 (JL is Kleisli). *Let $\mathcal{X}$ and $\mathcal{Y}$ be GPAs. A JL-simulation R from $\mathcal{X}$ to $\mathcal{Y}$ induces a fwd.-bwd. (Kleisli) simulation from $\mathcal{X}$ to $\mathcal{Y}$.* □

5.6 Generic Trace Semantics

Like in [16], the principal aim of Kleisli simulation is to show *trace inclusion*—a refinement relation with respect to (linear time) *trace semantics* which is the coarsest in the spectrum of [3]. Our use of the generic notion of Kleisli coalgebra calls for a generic definition of trace semantics too. We employ the theory in [8]; here is its quick recap.

A polynomial functor F always has an *initial algebra* $\alpha : FA \xrightarrow{\cong} A$. The intuition is: F represents a set of datatype constructors; and A is the induced inductive datatype. The algebraic structure α always becomes an invertible function.

Example 5.17. The functor $F = \{\checkmark\} + \mathsf{Ac} \times (\_)$ is thought of as: a nullary constructor $\checkmark$ and a family of unary constructors $a(\_)$, for each $a \in \mathsf{Ac}$. The induced inductive datatype is the set $\mathsf{Ac}^* = \{a_1 a_2 \cdots a_n \checkmark \mid n < \omega, a_i \in \mathsf{Ac}\}$ of (finite) lists over Ac. This set indeed carries an initial algebra: there is a canonical algebraic structure $\alpha : \{\checkmark\} + \mathsf{Ac} \times \mathsf{Ac}^* \xrightarrow{\cong} \mathsf{Ac}^*$, namely

$$\checkmark \longmapsto \checkmark \text{ (the empty sequence)}, \qquad (a, a_1 \cdots a_n \checkmark) \longmapsto a a_1 \cdots a_n \checkmark \enspace .$$

The following is the main result in [8], adapted to the current context.

Theorem 5.18 (Generic (finite) trace semantics). *Let $\alpha : FA \to A$ be an initial algebra. Given any F-system $\mathcal{X} = (X, s, c)$, there is a unique Kleisli arrow $\mathsf{tr}(c)$ that makes the diagram on the right commute. In particular, the Kleisli coalgebra $J(\alpha^{-1})$ is a* final *coalgebra.*

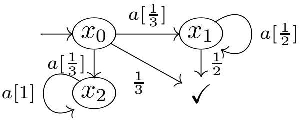
$$\begin{tikzcd}
FX \arrow[rr, dashed, "F(\mathsf{tr}(c))"] & & FA \\
X \arrow[u, "c"] \arrow[rr, dashed, "\mathsf{tr}(c)"] & & A \arrow[u, "J(\alpha^{-1})"'] \\
& \{*\} \arrow[ul, bend left, "s"] \arrow[ur, bend right, "\mathsf{tr}(c) \odot s"'] &
\end{tikzcd} \tag{7}$$

We set $\mathsf{tr}(\mathcal{X}) := \mathsf{tr}(c) \odot s$. It is $\mathsf{tr}(\mathcal{X}) : \{*\} \to \mathcal{D}A$ as a function, hence is a subdistribution over A. This $\mathsf{tr}(\mathcal{X})$ is referred to as the *(finite) trace semantics* of $\mathcal{X}$. To summarize: the *action type* F determines the set A of *linear-time behavior*; $\mathcal{X}$'s trace semantics $\mathsf{tr}(\mathcal{X})$ tells us which linear-time behavior is exhibited with how much likelihood.

Example 5.19 (Trace semantics for GPA). Let $F = \{\checkmark\} + \mathsf{Ac} \times (\_)$. The diagram (7) translates into the following conditions, where σ ranges over Ac^*.

$$\begin{aligned} \mathsf{tr}(c)(x)(\checkmark) &= \Pr[x \to \checkmark] \ , \\ \mathsf{tr}(c)(x)(a\sigma) &= \textstyle\sum_{x' \in X} \Pr[x \xrightarrow{a} x'] \cdot \mathsf{tr}(c)(x')(\sigma) \ ; \quad \text{and} \\ \mathsf{tr}(\mathcal{X})(\sigma) &= \mathsf{tr}(c)(x_0)(\sigma) \quad \text{when } s(*) = [x_0 \mapsto 1]. \end{aligned}$$

This is a reasonable definition of a "trace semantics" for GPA; resulting is a subdistribution $\mathsf{tr}(\mathcal{X})$ over lists on Ac. For example, let $\mathcal{X}$ be the GPA below on the left; then its trace semantics is as on the right.



$$\mathsf{tr}(\mathcal{X}) = \begin{bmatrix} \checkmark \mapsto \frac{1}{3},\ a\checkmark \mapsto \frac{1}{3} \cdot \frac{1}{2},\ a^2\checkmark \mapsto \frac{1}{3} \cdot \frac{1}{2} \cdot \frac{1}{2},\ \cdots \\ a^n\checkmark \mapsto \frac{1}{3} \cdot \left(\frac{1}{2}\right)^n,\ \cdots \end{bmatrix}$$

Note that our trace semantics only captures *finite* behavior; infinite sequences like a^ω are not in its domain Ac^*. With infinite behavior included we no longer have a clean characterization like in Thm. 5.18.

Like the definition of Kleisli simulation, the generic trace semantics (Thm. 5.18) also applies to other kinds of branching such as non-determinism. See [8].

5.7 Soundness Theorems

We recall the soundness result [5] for Kleisli simulation, via which soundness of JL- and HJ-simulation immediately follows. Its short proof in [5] makes use of order-theoretic properties of the diagram (7).

Theorem 5.20 (Soundness of Kleisli). *Let $\mathcal{X}, \mathcal{Y}$ be F-systems. Existence of a Kleisli simulation from $\mathcal{X}$ to $\mathcal{Y}$ implies trace inclusion:* $\mathsf{tr}(\mathcal{X}) \sqsubseteq \mathsf{tr}(\mathcal{Y})$. *Here a Kleisli simulation can be any of forward, backward, or hybrid.* □

Using Thm. 5.15 and Cor. 5.16, we immediately obtain soundness of JL-simulation (Def. 4.12). This is new to the best of the author's knowledge. Therefore the notion of JL-simulation can also be used for proving trace inclusion between GPAs, a use that has not been investigated much in the literature. The same applies to JL- and JL'-simulation for DTMCs; we postpone detailed treatment to another venue.

6 Conclusions and Future Work

We have showed that JL-simulation is a special case of HJ-simulation, which is further a special case of Kleisli simulation. This allows to transfer general results for a latter notion to a former one, most notably soundness.

Finding a Kleisli simulation is reduced to solving a family of linear inequalities. Its algorithmic aspect is to be investigated. We also aim to exploit acquired genericity and apply our results to other kinds of systems. We are interested in *stochastic context-free grammars* which have their application in modeling the secondary structure of RNA [2].

References

1. Baier, C., Katoen, J.P., Hermanns, H., Wolf, V.: Comparative branching-time semantics for markov chains. Inf. & Comp. 200(2), 149–214 (2005)
2. Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G.: Biological Sequence Analysis. Cambridge Univ. Press, Cambridge (1998)
3. van Glabbeek, R.J.: The linear time–branching time spectrum I; the semantics of concrete, sequential processes. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, ch. 1, pp. 3–99. Elsevier, Amsterdam (2001)
4. van Glabbeek, R.J., Smolka, S.A., Steffen, B.: Reactive, generative, and stratified models of probabilistic processes. Inf. & Comp. 121, 59–80 (1995)
5. Hasuo, I.: Generic forward and backward simulations. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 406–420. Springer, Heidelberg (2006)
6. Hasuo, I.: Tracing Anonymity with Coalgebras. PhD thesis, Radboud Univ. Nijmegen (2008)
7. Hasuo, I.: Generic forward and backward simulations II: Probabilistic simulations. To appear in RIMS Preprints (June 2010) (Extended version)
8. Hasuo, I., Jacobs, B., Sokolova, A.: Generic trace semantics via coinduction. Logical Methods in Comp. Sci. 3(4:11) (2007)
9. Hasuo, I., Kawabe, Y., Sakurada, H.: Probabilistic anonymity via coalgebraic simulations. Theor. Comp. Sci. 411(22-24), 2239–2259 (2010)
10. Hughes, J., Jacobs, B.: Simulations in coalgebra. Theor. Comp. Sci. 327(1-2), 71–108 (2004)
11. Jacobs, B.: Introduction to coalgebra. In: Towards mathematics of states and observations, Draft of a book (2005), www.cs.ru.nl/B.Jacobs/PAPERS
12. Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: LICS, pp. 266–277. IEEE Computer Society, Los Alamitos (1991)
13. Klin, B.: Bialgebraic methods and modal logic in structural operational semantics. Inf. & Comp. 207(2), 237–257 (2009)
14. Kulkarni, V.G.: Modeling and Analysis of Stochastic Systems. Chapman & Hall, Boca Raton (1995)
15. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Inf. & Comp. 94(1), 1–28 (1991)
16. Lynch, N., Vaandrager, F.: Forward and backward simulations. I. Untimed systems. Inf. & Comp. 121(2), 214–233 (1995)
17. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theor. Comp. Sci. 249, 3–80 (2000)
18. Segala, R.: Modeling and verification of randomized distributed real-time systems. PhD thesis. MIT, Cambridge (1995)
19. Sokolova, A.: Coalgebraic Analysis of Probabilistic Systems. PhD thesis, Techn. Univ. Eindhoven (2005)

Kleene, Rabin, and Scott Are Available

Jochen Hoenicke[1], Roland Meyer[2], and Ernst-Rüdiger Olderog[3,⋆]

[1] University of Freiburg
hoenicke@informatik.uni-freiburg.de
[2] LIAFA, Paris Diderot University & CNRS
roland.meyer@liafa.jussieu.fr
[3] University of Oldenburg
olderog@informatik.uni-oldenburg.de

Abstract. We are concerned with the availability of systems, defined as the ratio between time of correct functioning and uptime. We propose to model guaranteed availability in terms of *regular availability expressions* (rae) and *availability automata*. We prove that the intersection problem of rae is undecidable. We establish a Kleene theorem that shows the equivalence of the formalisms and states precise correspondence of flat rae and simple availability automata. For these automata, we provide an extension of the powerset construction for finite automata due to Rabin and Scott. As a consequence, we can state a complementation algorithm. This enables us to solve the synthesis problem and to reduce model checking of availability properties to reachability.

1 Introduction

Traditional approaches to system verification rely on idealistic assumptions, e.g., that each component will work perfectly all the time. However, in many applications such assumptions are unrealistic. Think of a sensor network where some of the sensors fail and recover. Is then the whole information accumulated by the network invalid?

Our paper is motivated by the desire to establish correctness properties of unreliable reactive systems where components may fail for some time, or phrased positively, are available only for a certain amount of time during an observation interval. This property is known as *(interval) availability.* It is often studied in the context of stochastic systems where one calculates the *probability* that a component or system has a certain interval availability but it may also be studied in the context of timed systems [dSeSG89, RS93, Tri01]. For continuous time models, availability can be formalised using integrals. Letting $sys(t)$ represent proper system functionality ($\{0, 1\}$-valued) at time t, the expression

$$\frac{1}{n} \cdot \int_0^n sys(t)\; dt \geq k$$

[⋆] This work was partly supported by the German Research Council (DFG) under the grant SFB/TR 14 AVACS. The second author was supported by the French ANR projects Averiss and Veridyc.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 462–477, 2010.

states that the *ratio* of accumulated time the system is functioning as desired to total uptime n is at least $k \in [0, 1]$. Thus, during the observation interval $[0, n]$ the system is available for fraction k of the time.

We discovered that availability can be studied already in the simpler setting of discrete time, in terms of formal languages and automata-theoretic models. This is what this paper is about. We express availability as the ratio of letters from a set A in a word $w = a_1 \cdots a_n$ to the length n of the word. Formula

$$\frac{1}{n} \cdot \sum_{t=1}^{n} \chi_A(a_t) \geq k$$

states that in word w the letters of the set A are available for at least fraction k of the time. When modelling, the set A may contain desired system states or receipt actions of messages. Our contributions are as follows:

1. We introduce (in Section 2) the class of *regular availability expressions* (*rae* for short). We prove that the problem whether the intersection of finitely many raes denotes the empty language is *undecidable*. The proof is by reduction of the termination problem of Minsky machines.
2. We introduce (in Section 3) *availability automata* and establish a *Kleene theorem*. It states that the class of languages denoted by flat raes coincides with the class accepted by *simple* availability automata. We derive a correspondence between intersections of raes and availability automata.
3. We extend (in Section 4) the *powerset construction* for finite automata [RS59]. For a nondeterministic *simple* availability automaton it computes an equivalent deterministic version. As a consequence, we derive a *complementation* algorithm for simple automata that solves the *synthesis* problem and reduces *model checking* of availability properties to reachability.

2 Regular Availability Expression

We define availability for *finite words* $w \in \Sigma^*$ over an alphabet Σ. We denote by $|w|$ the length of the word. For $A \subseteq \Sigma$ we denote by $\pi_A(w)$ the projection of w to the alphabet A, i. e., the word that is derived from w by removing all letters that are not in A. This notation allows for a concise definition of availability that avoids division by zero in case $w = \varepsilon$:

$$\frac{1}{n} \cdot \sum_{i=1}^{n} \chi_A(a_i) \geq k \quad \text{ iff } \quad |\pi_A(w)| \geq k|w| \quad \text{with } w = a_1 \dots a_n.$$

Definition 1 (Syntax of raes). *The set of* regular availability expressions (rae) over an alphabet Σ *is inductively defined as follows:*

$$rae ::= a \mid rae + rae \mid rae.rae \mid rae^* \mid \checkmark \mid rae_{A \gtrsim k}$$

with $a \in \Sigma$, $A \subseteq \Sigma$, $\gtrsim\, \in \{\geq, >\}$, *and* $k \in [0, 1]$.

The symbol ✓ marks the positions at which the required availability is checked. At these positions, the expression $rae_{A \geq k}$ ensures that the letters in A are available for at least fraction k. A highly available network may be specified by the expression $((up + down)^*.\checkmark)_{\{up\} > 0.99}$. We abbreviate $rae_{(\Sigma \setminus A) \geq 1-k}$ by $rae_{A \leq k}$.

An rae is *flat* if the operator $rae_{A \gtrsim k}$ does not appear nested. Flat raes are the intuitive model one expects. The system behaviour is captured by a regular expression. For analysis purposes, availability constraints are added to restrict the traces to the average behaviour. As the system typically does not react to the occurrence number of events, nesting of availability operators is hardly needed.

Raes are given a semantics in terms of languages $\mathcal{L}\,(rae) \subseteq (\Sigma \cup \{\checkmark\})^*$:

$$\begin{aligned} \mathcal{L}\,(a) &:= \{a\} & \mathcal{L}\,(rae_1 + rae_2) &:= \mathcal{L}\,(rae_1) \cup \mathcal{L}\,(rae_2) \\ \mathcal{L}\,(rae^*) &:= \mathcal{L}\,(rae)^* & \mathcal{L}\,(rae_1.rae_2) &:= \mathcal{L}\,(rae_1)\,.\mathcal{L}\,(rae_2) \\ \mathcal{L}\,(\checkmark) &:= \{\checkmark\} & \mathcal{L}\left(rae_{A \gtrsim k}\right) &:= \mathcal{L}\,(rae)_{A \gtrsim k}\,. \end{aligned}$$

Operator $\mathcal{L}_{A \gtrsim k}$ collects all words in language $\mathcal{L}$, where each prefix ending in ✓ satisfies the availability $A \gtrsim k$ as discussed above (not counting ✓-symbols). The operator also removes these symbols. Formally,

$$\mathcal{L}_{A \gtrsim k} := \{\pi_\Sigma(w) \mid w \in \mathcal{L} \text{ and } |\pi_A(w_1)| \gtrsim k|\pi_\Sigma(w_1)| \text{ for all } w_1.\checkmark.w_2 = w\}.$$

Raes and classical regular expressions differ in their properties. Raes have an undecidable language *intersection problem*

$$\mathcal{L}\,(rae_1) \cap \mathcal{L}\,(rae_2) = \emptyset. \qquad (\textbf{Intersect})$$

Theorem 1. Intersect *is undecidable.*

Proof. The proof of Theorem 1 is by reduction of the termination problem for Minsky machines to the intersection problem for raes. Since Minsky machines are Turing complete [Min67, Theorem 14.1-1], termination is undecidable.

A Minsky machine $M = (c_1, c_2, inst)$ has two *counters* c_1 and c_2 that store arbitrarily large natural numbers and a *finite set inst of labelled instructions* $l : op$. There are two kinds of operations op. The first, denoted by $inc(c, l')$, increments counter $c \in \{c_1, c_2\}$ by one and then jumps to the instruction labelled by l'. The second command, denoted by $dect(c, l', l'')$, is called a *decrement and test*. It checks counter c for being zero and, in this case, jumps to instruction l'. If the value of c is positive, the counter is decremented and the machine jumps to l''. We use $Locs(inst) := \{l \mid l : op \in inst\}$ to refer to the *set of control locations* in M. It contains an *initial instruction* $l_I \in Locs(inst)$ that starts the computation of the Minsky machine. A *final label* $l_F \notin Locs(inst)$ may appear only as the destination of an instruction and terminates the computation.

Given a Minsky machine $M = (c_1, c_2, inst)$ we define two raes so that the intersection of their languages is non-empty iff the computation of M terminates.

Construction. We start by splitting each instruction in *inst* into two parts, one part for each counter. This yields two new sets of instructions $inst_1$ and $inst_2$. The parts added to $inst_1$ only affect counter c_1 and jump to the second part of the instruction in inst_2. The parts added to inst_2 affect counter c_2 and jump to the first part of the next instruction. Since every instruction in inst only changes one counter, we need a new operation $\mathit{goto}(l)$ to jump when no change is made:

$$\begin{aligned} l_i : \mathit{inc}(c_1, l_j) \in \mathit{inst} \quad &\leadsto \quad l_i : \mathit{inc}(l'_i) \in \mathit{inst}_1 \wedge l'_i : \mathit{goto}(l_j) \in \mathit{inst}_2 \\ l_i : \mathit{dect}(c_1, l_j, l_k) \in \mathit{inst} \quad &\leadsto \quad l_i : \mathit{dect}(l'_i, l''_i) \in \mathit{inst}_1 \wedge l'_i : \mathit{goto}(l_j) \in \mathit{inst}_2 \\ &\qquad\qquad\qquad\qquad\qquad \wedge\, l''_i : \mathit{goto}(l_k) \in \mathit{inst}_2 . \end{aligned}$$

The translation for the second counter is similar. By this transformation, two adjacent instructions $l : \mathit{op}$, $l' : \mathit{op}$ always change first the counter c_1 and then c_2. We encode the computation steps of the machines as $l.c^v.a.r^{v'}.\ l'.c^u.b.r^{u'}$. The sequence of $v \in \mathbb{N}$ letters c encodes the valuation of the counter c_1 before the first instruction. The following symbol a is either i, d, or ε depending on whether operation $l : \mathit{op}$ increments the first counter, decrements it, or does not act on it. An ε is also used if op tests the first counter for being zero. The result of the operation is stored as a sequence of result letters $r^{v'}$. We ensure that $v' = v + 1$ if a is an increment (similarly for decrement and goto) by an availability expression *av1*, indicated by a brace in Formula 1 below. Then the operation on c_2 starts, indicated by its label l'. In the encoding of the second counter, b represents the operation to be performed.

The crucial issue is to transfer the result $r^{v'}$ of an operation to the next instruction that is executed. Again, we use an availability expression *av2*, which now connects the encodings of two subsequent computation steps. To sum up, a terminating computation of the machine is encoded by the word

$$\big[l_I.(\rlap{\overbrace{c^{v_0}.a_0.r^{v_1}}^{\mathit{av1}}}\hphantom{c^{v_0}.a_0.}\underbrace{r^{v_1}).l'_I.(c^{u_0}.b_0.r^{u_1})\big].\quad \big[l_1.(c^{v_1}}_{\mathit{av2}}.a_1.r^{v_2}).l'_1.(c^{u_1}.b_1.r^{u_2})\big]\ldots l_F . \qquad (1)$$

To encode the effect of a *goto*, we demand equality between the number of c and r symbols by stating that the availability of c is precisely $^1/_2$. This is achieved by the regular availability expression

$$\mathit{GOTO} := \big((c^*.r^*.\checkmark)_{\{c\}\geq {}^1/_2}.\checkmark\big)_{\{c\}\leq {}^1/_2}\,.$$

To avoid clutter we abbreviate this by $(c^*.r^*.\checkmark)_{\{c\}={}^1/_2}$. Note that this trick is only valid if there is exactly one $\checkmark$-symbol at the end of the expression. The following availability expressions implement increment and decrement operations:

$$\mathit{INC} := (c^*.i.r^*.\checkmark)_{\{c,i\}={}^1/_2} \qquad\qquad \mathit{DEC} := (c^*.d.r^*.\checkmark)_{\{c,i\}={}^1/_2}.$$

In the increment expression, the symbol i is counted like c and has to match an additional r symbol, which ensures that the number of r symbols is by one larger than the number of c symbols. Likewise in the decrement expression, the symbol d has to match an additional c symbol, so the result value encoded by the number of r symbols is by one smaller than the number of c symbols. Thus all expressions encoding operations have the shape

$$(c^*.\mathit{CMD}.r^*.\checkmark)_{\{c,i\}={}^1/_2} \qquad \text{with } \mathit{CMD} := i + d + \varepsilon.$$

With these definition, we define an availability expression $rae(l : op)$ for every instruction $l : op$. It ensures correct computation and control flow. Note that tests for zero need to be implemented by ε instead of $GOTO$:

$$rae(l : inc(l')) := l.INC.l' \qquad rae(l : dect(l', l'')) := l.l' + l.DEC.l''$$
$$rae(l : goto(l')) := l.GOTO.l'.$$

Combining all commands on a counter, we define

$$OP_i := \Sigma_{l:op\in inst_i} rae(l : op) \quad \text{for } i = 1, 2.$$

We rely on two availability expressions that define a word corresponding to a terminating execution of the Minsky machine. The first expression imitates instructions on the first counter and copies contents of the second. It also ensures that the execution ends at the final label and that the second counter is initialised to zero. The second expression executes instructions on the second counter and copies the contents of the first. It also starts the execution at the initial label and initialises the first counter with zero:

$$rae_{cmp1}(M) := OP_1.\varepsilon.CMD.\left[(r^*.OP_1.c^*.\checkmark)_{\{c,i\}\cup Locs(inst_1)={}^1\!/_2}.CMD\right]^*.r^*.l_F$$
$$rae_{cmp2}(M) := l_I.\varepsilon.CMD.\left[(r^*.OP_2.c^*.\checkmark)_{\{c,i\}\cup Locs(inst_1)={}^1\!/_2}.CMD\right]^*.r^*.OP2.$$

To copy the result value r^* from one computation step to the counter value c^* in the next, we again employ availability expressions, i.e., we implement *av2* in Formula 1. One difficulty is that this copy operation is interrupted by an OP. However, the definitions of INC, DEC, and $GOTO$ guarantee an availability of ${}^1\!/_2$ for the letters c, i between the labels. There are two labels in OP; since those from $Locs(inst_1)$ are in the availability set, the full OP-command guarantees an availability of exactly ${}^1\!/_2$. As a result, we obtain equality between r^* and c^*.

One can show that M terminates if and only if the intersection

$$\mathcal{L}\left(rae_{cmp1}(M)\right) \cap \mathcal{L}\left(rae_{cmp2}(M)\right) \neq \emptyset.$$

is non-empty. The details can be found in the appendix. □

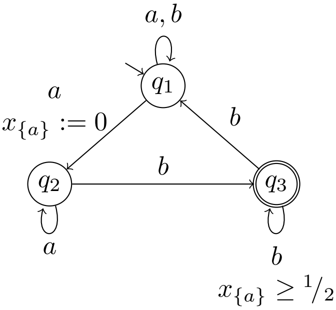
Remark 1. The intersection problem remains undecidable for eight flat raes. Each of the raes in the previous construction can be expressed by four flat versions (note that an equality requires two constraints).

Theorem 1 shows that raes correspond to an automaton model that is strictly more expressive than finite automata. We investigate it in the following section.

3 Availability Automata

We define *availability automata* as transition labelled finite automata enriched by *(availability) counters* that determine the presence of certain letters within a run. Each counter represents a *check* operation of an availability $A \gtrsim k$ with $A \subseteq \Sigma$ and $k \in [0, 1]$, and transitions in the availability automaton are guarded by these constraints. Additionally, each transition carries a *reset* operation $Y := 0$ that denotes the counters that are reset so as to restart the measurement afterwards.

Definition 2 (Availability automata). *Let Σ be an alphabet. An* availability automaton *over Σ is a tuple $\mathcal{A} = (Q, Q_I, Q_F, X, \rightarrow, c)$ with* states Q, initial states $Q_I \subseteq Q$, *and* final states $Q_F \subseteq Q$. *The* availability counters *are given by X. The* counter labelling function $c : X \rightarrow (\mathbb{P}(\Sigma) \times \{\geq, >\} \times [0, 1])$ *assigns to each counter $x \in X$ a constraint $A \gtrsim k$.* Transitions $\rightarrow \subseteq Q \times \Sigma \times \mathbb{P}(X) \times \mathbb{P}(X) \times Q$ *are labelled by Σ, check the constraints of their counters, and reset some counters.*



Consider the availability automaton to the right. We employ the following notation. States are drawn as nodes. Initial states have an incoming arc, final states carry a double circle. The first part $A \subseteq \Sigma$ of the counter labelling $c(x) = (A \gtrsim k)$ is given as index x_A of the counter. If two counters are indexed by the same set A, we use different variables x_A and y_A. A transition $(q_1, a, C, Y, q_2) \in \rightarrow$ is drawn as directed arc from q_1 to q_2 labelled by a. A check operation $C = \{x_A\}$ is written as $x_A \gtrsim k$, revealing the remaining part of the counter labelling. A reset set $Y = \{x_A\}$ is denoted by $x_A := 0$.

To give an operational semantics, we exploit the following equivalence that highlights the contribution of a single action to an availability expression:

$$|\pi_A(w)| \gtrsim k|w| \quad \text{iff} \quad |\pi_A(w)| - k|w| \gtrsim 0.$$

To check the availability $A \gtrsim k$, we compare the value of $|\pi_A(w)| - k|w|$ against zero. More importantly, this value can be computed incrementally by adding 1 for each occurrence of a symbol of A and subtracting k for every symbol. The observation suggests the use of *counter valuations* $\gamma : X \rightarrow \mathbb{R}$. They assign to each counter the value $|\pi_A(w)| - k|w|$ of the word w, which has been read since the last reset. For $a \in \Sigma$, we denote by $\chi(a) : X \rightarrow \mathbb{R}$ the counter valuation that assigns $\chi_A(a) - k$ to counter x checking $c(x) = A \gtrsim k$. As usual, χ_A is the characteristic function. This enables us to describe the *update* when processing the character a by $\gamma' = \gamma + \chi(a)$. The *reset* $\gamma' = \gamma[Y := 0]$ yields $\gamma'(x) = 0$ if $x \in Y$ and $\gamma'(x) = \gamma(x)$ otherwise.

The semantics of an availability automaton $\mathcal{A} = (Q, Q_I, Q_F, X, \rightarrow, l)$ is given in terms of runs in the set $\mathcal{R}(\mathcal{A})$. A *run* r is a sequence

$$r = q_0.\gamma_0.a_1.q_1.\gamma_1.a_2.q_2.\gamma_2 \ldots a_n.q_n.\gamma_n \in \mathcal{R}(\mathcal{A})$$

subject to the following constraints. Initially, all counters are zero, $\gamma_0(x) = 0$ for all $x \in X$, and the run starts in an initial state, $q_0 \in Q_I$. For every step $q_{i-1}.\gamma_{i-1}.a_i.q_i.\gamma_i$ there is a transition $(q_{i-1}, a_i, C, Y, q_i) \in \rightarrow$ such that $\gamma_i = (\gamma_{i-1} + \chi(a_i))[Y := 0]$ and for each constraint $x \in C$, $\gamma_{i-1}(x) + \chi(a_i)(x) \gtrsim 0$. By the above transformation, this guarantees the desired availability.

Runs contain internal information about states and counter valuations. We abstract them away to obtain the usual notion of the language of an automaton.

Definition 3 (Language). *The* language of $\mathcal{A} = (Q, Q_I, Q_F, X, \rightarrow, l)$ *is the projection of all runs that end in a final location to their labels*

$$\mathcal{L}(\mathcal{A}) := \{a_1 \dots a_n \mid q_0.\gamma_0.a_1.q_1.\gamma_1 \dots a_n.q_n.\gamma_n \in \mathcal{R}(\mathcal{A}) \text{ with } q_n \in Q_F\}.$$

Like for finite automata, ε-transitions do not contribute to the expressiveness of availability automata but are convenient when encoding raes.

Lemma 1. *For every $\Sigma \cup \{\varepsilon\}$-labelled availability automaton $\mathcal{A}$, there is a Σ-labelled availability automaton $\mathcal{A}'$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

As we will show later, the language emptiness problem is undecidable for general availability automata. However, for automata with only one active counter in each state emptiness becomes decidable. We call them *simple.*

Definition 4 (Simple availability automata). *An availability automaton $\mathcal{A} = (Q, Q_I, Q_F, X, \rightarrow, c)$ is called* simple, *if there is at most one active counter in each state. Formally, there is a mapping $\mu : Q \rightarrow X$ such that each transition $(q, a, C, Y, q') \in \rightarrow$ is only constrained by the counter of q, $C \subseteq \{\mu(q)\}$, and resets the counter of q' if it differs from the counter of q, $\{\mu_{\mathcal{A}}(q')\} \subseteq \{\mu_{\mathcal{A}}(q)\} \cup Y$.*

Simple automata capture precisely the languages of flat raes and are thus of particular interest. The following two sections are devoted to the proof of this statement. In general, availability automata are equivalent to intersections of raes — modulo renaming.

3.1 From Expressions to Automata

To encode raes into availability automata, we define operators on the automata that mimic the operations on raes. *Choice* $\mathcal{A}_1 + \mathcal{A}_2$, *sequential composition* $\mathcal{A}_1.\mathcal{A}_2$, and *iteration* $\mathcal{A}^*$ correspond to the constructions for finite automata. Choice is defined by union, sequential composition introduces ε-transitions from the final states of the first to the initial states of the second automaton, and iteration introduces ε-transitions back to a fresh initial and final state. Different from the classical definitions, counters are reset when a new automaton is entered in $\mathcal{A}_1.\mathcal{A}_2$ and $\mathcal{A}^*$. All these operations produce simple availability automata if the input automata are simple.

An *availability constraint* $\mathcal{A}_{A \gtrsim k}$ is reflected by a relabelling of the automaton. The idea is to let every former $\surd$-labelled transition check the constraint $A \gtrsim k$. To this end, a new counter x_A is added to the automaton and every $\surd$-transition is relabelled to ε and augmented by the counter's constraint $x_A \gtrsim k$. For flat raes, the operation results in a simple automaton, since the input is a finite automaton without counters.

To reflect the intersection of languages — an essential ingredient in automata-theoretic verification procedures — we define a *synchronous product* $\mathcal{A}_1 \parallel \mathcal{A}_2$. It multiplies the states $Q^1 \times Q^2$, takes the pairs of initial states as initial $Q^1_I \times Q^2_I$, and likewise as final states $Q^1_F \times Q^2_F$. Transitions synchronise on the label and combine the guards. Note that simple availability automata are not closed under synchronous product. The operators reflect their semantics counterparts.

Proposition 1 (Semantic correspondence)

$$\mathcal{L}(\mathcal{A}_1.\mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1).\mathcal{L}(\mathcal{A}_2) \qquad \mathcal{L}(\mathcal{A}^*) = \mathcal{L}(\mathcal{A})^*$$
$$\mathcal{L}(\mathcal{A}_1 + \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2) \qquad \mathcal{L}\left(\mathcal{A}_{A \gtrsim m}\right) = \mathcal{L}(\mathcal{A})_{A \gtrsim m}$$
$$\mathcal{L}(\mathcal{A}_1 \parallel \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2).$$

Proposition 1 paves the way for a compositional definition of the automaton representation $\mathcal{A}[\![rae]\!]$ of an rae. A single action a is translated to the automaton that has an a-transition from an initial to a final state. The remaining operators are replaced homomorphically by their automata-theoretic counterparts.

Proposition 2 (Kleene's first half). $\mathcal{L}(rae) = \mathcal{L}(\mathcal{A}[\![rae]\!])$. *Moreover, if rae is flat, then* $\mathcal{A}[\![rae]\!]$ *is simple.*

3.2 From Automata to Expressions

We start with a simple availability automaton $\mathcal{A} = (Q, Q_I, Q_F, X, \rightarrow, c)$ and its counter mapping $\mu : Q \rightarrow X$. Without changing the accepted language, we can strip resets of counters that are not active in the successor state. We compute a corresponding flat rae in two steps. First, we compute $rae_{q,q'}$ that describes all words from q to q' that obey the counter's availability constraint. More precisely, the rae models all words that are accepted by $\mathcal{A}$ when starting in q with a zero counter and reaching q' with a reset only on the last edge. Therefore, the concatenation $rae_{q,q'}.rae_{q',q''}$ again corresponds to some partial run of $\mathcal{A}$. Likewise, we determine availability expressions rae_q for the language from q (with a zero counter and without passing reset edges) to a final state.

In a second step, we construct a finite state automaton which has these raes as transition labels. When using Kleene's classical result to compute the regular expression corresponding to the automaton, we obtain a flat rae. It has precisely the language of $\mathcal{A}$. Since we recorded the measurements as raes, there is no need to add further availability constraints.

Phase 1. For every pair of states $q, q' \in \mathcal{A}$, we construct a finite automaton[1] $A_{q,q'}$. We take the graph of $\mathcal{A}$, make q the initial state, and add a fresh final state. Resetting transitions that previously ended in q' are redirected to the final state. Thus, the measurement between q and q' ends with a reset. The remaining resetting transitions are removed. Since $\mathcal{A}$ is simple, this makes all states with a different active counter unreachable, so they can be removed together with their outgoing edges. The resulting automaton has the single counter $x := \mu(q)$. To reflect measurements of $c(x)$ on an a-labelled transition, we split the edge. The first new transition is labelled by a, the second by $\checkmark$. Let $\rightarrow_x$ denote the outgoing transitions from a state where x is active, transitions $\rightarrow_{res} \subseteq \rightarrow_x$ reset the counter, and $\rightarrow_{c(x)} \subseteq \rightarrow_x$ have $c(x)$ as guard. By $\rightarrow_{std}$ we refer to unguarded

[1] A finite automaton is a tuple $A = (Q, Q_I, Q_F, \rightarrow)$ with the typical interpretation as *states*, *initial states*, *final states*, and *transition relation* $\rightarrow \subseteq Q \times \Sigma \cup \{\varepsilon, \checkmark\} \times Q$.

and reset-free transitions in $\rightarrow_x \setminus (\rightarrow_{res} \cup \rightarrow_{c(x)})$. They can be understood as transitions of a finite automaton:

$$A_{q,q'} := (\mu^{-1}(x) \cup \{q_e \mid e \in \; \rightarrow_{c(x)}\} \cup \{q_\nu\}, \{q\}, \{q_\nu\}, \rightarrow_{std} \cup \rightarrow_{split} \cup \rightarrow_{redir}),$$

with $\rightarrow_{split} := \{(p, a, q_e), (q_e, \checkmark, p') \mid e = (p, a, \{x\}, \emptyset, p') \in \; \rightarrow_{c(x)}\}$ and $\rightarrow_{redir} := \{(p, a, q_\nu) \mid (p, a, \emptyset, \{\mu(q')\}, q') \in \; \rightarrow_{res}\}$. If the resetting transition to q' is guarded, also the redirected edge is split up.

The finite automaton A_q is constructed similarly. It has q as initial state, uses the final states of the original automaton, and removes all resetting edges. Guarded transitions are again split up and decorated by $\checkmark$:

$$A_q := (\mu^{-1}(x) \cup \{q_e \mid e \in \; \rightarrow_{c(x)}\}, \{q\}, Q_F, \rightarrow_{std} \cup \rightarrow_{split})$$

We compute an ordinary regular expression $re_{q,q'}$ that accepts the language of $A_{q,q'}$. Adding the constraint $c(x)$ yields $rae_{q,q'} := [re_{q,q'}]_{c(x)}$ that reflects the measurement between q and q'. We also construct re_q and define $rae_q := [re_q]_{c(x)}$.

Phase 2. From $\mathcal{A}$ we construct a finite automaton A that describes its language. Again, we preserve the states, keep the initial states, and add a fresh final state. The main idea is to summarise all paths between two states q and q' by a single transition that is labelled by $rae_{q,q'}$. The expression takes care of the required availability constraint. Similarly, from every state q we have a transition to the new final state that is labelled by rae_q:

$$A := (Q \cup \{q_\nu\}, Q_I, \{q_\nu\}, \{(q, rae_{q,q'}, q') \mid q, q' \in Q\} \cup \{(q, rae_q, q_\nu) \mid q \in Q\}).$$

Let $rae[\![\mathcal{A}]\!]$ denote the regular expression for A. Due to the flat raes as labels, the expression itself is again flat. It correctly represents the automaton's language.

Example 1. Applying the algorithm to the automaton from the beginning of Section 3 yields $((a+b)^*a)_{\{a\}\geq 1/2}((a^*b(b\checkmark)^*b(a+b)^*a)_{\{a\}\geq 1/2})^*(a^*b(b\checkmark)^*)_{\{a\}\geq 1/2}$.

Proposition 3 (Kleene's second half). $\mathcal{L}(\mathcal{A}) = \mathcal{L}(rae[\![\mathcal{A}]\!])$.

Proposition 2 and Proposition 3 establish our second main result. Flat regular availability expressions and simple availability automata are equally expressive.

Theorem 2 (Kleene theorem for availability). *A language is recognised by a flat rae if and only if it is accepted by a simple availability automaton.*

An availability automaton that is not simple can be decomposed into simple automata. Take a *free version* $\mathcal{A}_f$ of the automaton $\mathcal{A}$ where transitions e have unique labels, say a_e. The original language can be obtained by removing the indices with the homomorphism $h(a_e) = a$. Thus, $h(\mathcal{L}(\mathcal{A}_f)) = \mathcal{L}(\mathcal{A})$ holds. The free automaton $\mathcal{A}_f$ is decomposed into several simple availability automata $\mathcal{A}_f^x$ with x as single counter. This decomposition projects away the constraints on the other counters and leaves states and transitions unchanged.

Lemma 2. *Consider $\mathcal{A}_f$ with counters X. Then $\mathcal{L}(A_f) = \bigcap_{x \in X} \mathcal{L}\left(A_f^x\right)$.*

With the previous result we derive the following correspondence.

Corollary 1. *Up to a homomorphism, a language is recognised by an intersection of raes if and only if it is accepted by an availability automaton.*

4 A Powerset Construction

The well-known powerset construction of Rabin and Scott that constructs deterministic finite automata from non-deterministic ones can be extended to simple availability automata. Like for finite automata, the key idea is to record in the state of the deterministic automaton the possible states the non-deterministic automaton can be in, e.g. $\{p, q\}$ would be a state of the deterministic automaton. For availability automata, also the possible values of the active counter in different runs need to be represented. Therefore, the following observation is crucial to our construction. For simple automata, it is sufficient to record the highest availability for this counter. Thus, we record one value for each state of the original automaton, which can be achieved by $|Q|$ counters. For the state $\{p, q\}$, this yields counters x_p and x_q.

When the non-deterministic automaton changes its state without resetting the active counter, say from p to p', we need to set the counter $x_{p'}$ of the new state to the counter of the old state. As the syntax of availability automata does not allow for counter assignments, we use a mapping μ as part of the state of the deterministic automaton. It yields for each state in the current set of states the availability counter that stores the highest possible availability. In the running example, p' is assigned counter x_p. We shall also need to compare two counters. If p' is also reached from q, we need to know whether x_p is higher than x_q. Therefore, we keep an order $\succ$ on the counters as part of the deterministic state.

Given a simple availability automaton $\mathcal{A} = (Q, Q_I, Q_F, X, \rightarrow, c)$ with its active counter mapping $\mu_{\mathcal{A}} : Q \rightarrow X$, we construct an equivalent deterministic automaton $det(\mathcal{A})$. As argued, we need a counter for each state, $X^d = Q$. Additionally, we keep a copy of the counters $\overline{X}^d$. While x measures $A \gtrsim k$, counter $\bar{x}$ observes $A \lesssim k$. This allows for explicit checks of violations of an availability constraint. States Q^d of the deterministic automaton are triples $(\boldsymbol{q}, \mu, \succ)$ where

- $\boldsymbol{q} \subseteq Q$ is the set of possible states of the non-deterministic automaton like in the construction of Rabin and Scott.
- $\mu : \boldsymbol{q} \rightarrow X^d$ assigns to each possible state the counter that stores the corresponding availability. If a state $q \in \boldsymbol{q}$ is mapped to a counter $\mu(q) = x_{q'}$ corresponding to a state $q' \in Q$, we additionally require that q and q' have the same active counter $\mu_{\mathcal{A}}(q) = \mu_{\mathcal{A}}(q')$.
- $\succ \subseteq \mu(\boldsymbol{q}) \times \mu(\boldsymbol{q})$ is a order on the counters. First, the order is compatible with the counter valuations γ in a run, i.e., $y \succ x$ guarantees $\gamma(y) \geq \gamma(x)$. Additionally, $\succ$ is a linear order on those counters that correspond to the same counter in the original automaton $\mathcal{A}$. More precisely, for $q, q' \in \boldsymbol{q}$

with $\mu(q) \neq \mu(q')$, the corresponding counters are ordered ($\mu(q) \succ \mu(q')$ or $\mu(q') \succ \mu(q)$) if and only if the states q and q' have the same active counter $\mu_{\mathcal{A}}(q) = \mu_{\mathcal{A}}(q')$.

We define the powerset automaton as

$$det(\mathcal{A}) = (Q^d, Q^d_0, Q^d_F, X^d \cup \overline{X}^d, \rightarrow^d, c^d).$$

The initial state is $Q^d_0 = \{(Q_I, \mu_0, \succ_0)\}$ with the initial mapping μ_0 that assigns to each initial state $q \in Q_I$ the corresponding counter $x_q \in X^d$ and $\succ_0$ arbitrary. The final states are the states that contain a final state of the original automaton, $Q^d_F = \{(\boldsymbol{q}, \mu, \succ) \mid \boldsymbol{q} \cap Q_F \neq \emptyset\}$. The constraints c^d on the counters in X^d are taken from $\mathcal{A}$, $c^d(x_q) = c(\mu_{\mathcal{A}}(q))$ for $x_q \in X^d$. The counter $\overline{x}$ measures the inverse constraint. Since $A < k$ is equivalent to $\Sigma \setminus A > 1 - k$, we set $c^d(\overline{x}) = (\Sigma \setminus A > 1 - k)$ if $c^d(x) = (A \geq k)$, and similar for $A > k$.

In the deterministic automaton, the edges are labelled by checks C^d that contain for each $x \in X^d$ either x or $\overline{x}$. Furthermore, we require consistency of C^d with the order $\succ_1$ of the source location:

$$y \succ_1 x \quad \text{and} \quad x \in C^d \quad \text{implies} \quad y \in C^d.$$

Since the counters $x, \overline{x}$ measure opposite constraints and are reset at the same time, we have $\gamma(x) \gtrsim 0$ iff $\gamma(\overline{x}) \lesssim 0$ for every reachable counter valuation γ. This means at most one set of constraints C^d is satisfied by γ. Moreover, as was discussed, reachable γ are compatible with the order in the state. So, for $y \succ_1 x$ we obtain $\gamma(y) \geq \gamma(x)$ and thus $\gamma(x) \gtrsim 0$ implies $\gamma(y) \gtrsim 0$. Therefore, for each γ there is exactly one enabled C^d.

To define the transition relation $\rightarrow^d$ of the deterministic automaton we define the unique successor state $(\boldsymbol{q}_2, \mu_2, \succ_2)$ for each state $(\boldsymbol{q}_1, \mu_1, \succ_1)$, each symbol $a \in \Sigma$, and each constraint set C^d. The enabled transitions of the non-deterministic automaton are the outgoing a-labelled transitions from states in $\boldsymbol{q}_1$ for which the guard is true,

$$\begin{aligned} enabled := {} & \{(q_1, a, \emptyset, Y, q_2) \in \rightarrow \mid q_1 \in \boldsymbol{q}_1\} \\ & \cup \{(q_1, a, \{\mu_{\mathcal{A}}(q_1)\}, Y, q_2) \in \rightarrow \mid q_1 \in \boldsymbol{q}_1, \mu_1(q_1) \in C^d\}. \end{aligned}$$

The first part of the successor state $\boldsymbol{q}_2$ is computed as for Rabin and Scott. A state q_2 is in the set $\boldsymbol{q}_2$ if there is an enabled transition leading to it,

$$\boldsymbol{q}_2 := \{q_2 \in Q \mid \exists q_1, C, Y : (q_1, a, C, Y, q_2) \in enabled\}.$$

The difficult part is computing the new counter valuations for each state $q_2 \in \boldsymbol{q}_2$. If an enabled transition to q_2 resets the counter $\mu_{\mathcal{A}}(q_2)$ we may need a fresh counter in $det(\mathcal{A})$ that is reset on the transition. The counter of q_2 can also be inherited from a source location if there is an edge in the non-deterministic automaton that does not reset the counter. Let x_{q_2} denote a fresh counter for

q_2, then the new counter used as $\mu_2(q_2)$ is the counter with the highest value from the set of candidates

$$\begin{aligned} \mathit{cand}(q_2) := \{x_{q_2} \mid \exists q_1, C, Y : (q_1, a, C, Y, q_2) \in \mathit{enabled}, \mu_{\mathcal{A}}(q_2) \in Y\} \\ \cup \{\mu_1(q_1) \mid \exists C, Y : (q_1, a, C, Y, q_2) \in \mathit{enabled}, \mu_{\mathcal{A}}(q_2) \notin Y\}. \end{aligned}$$

The set $\mathit{cand}(q_2)$ is non-empty for all $q_2 \in \boldsymbol{q}_2$ and contains only counters x_q with $\mu_{\mathcal{A}}(q) = \mu_{\mathcal{A}}(q_2)$. The latter holds by definition of simple automata. A non-resetting transition from q_1 to q_2 requires the states to have the same active counter, $\mu_{\mathcal{A}}(q_1) = \mu_{\mathcal{A}}(q_2)$. The fresh counters x_{q_2} are inserted into the order $\succ_1$ based on the guard C^d. This yields a new order $\succ_2'$. For every state $q_1 \in \boldsymbol{q}_1$ with the same active counter in $\mathcal{A}$, we check whether $\mu_1(q_1)$ is positive, which means $\mu_1(q_1) \in C^d$. In this case, we add $\mu_1(q_1) \succ_2' x_{q_2}$, otherwise $x_{q_2} \succ_2' \mu_1(q_1)$.

The new counter for q_2 is the one from $\mathit{cand}(q_2)$ with the largest value,

$$\mu_2(q_2) := \max_{\succ_2'} \mathit{cand}(q_2) \quad \text{for } q_2 \in \boldsymbol{q}_2 .$$

This counter is well-defined since $\mathit{cand}(q_2)$ is a non-empty finite set on which $\succ_2'$ is a linear order. We obtain $\succ_2$ from $\succ_2'$ by removing all unused counters,

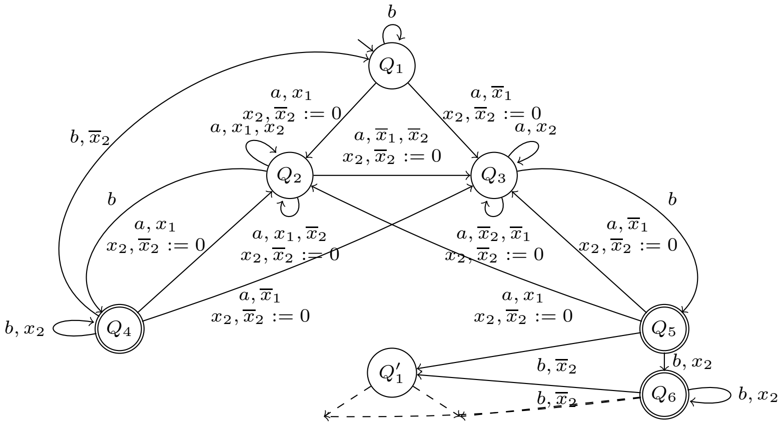
$$\succ_2 := \succ_2' \cap (\mu_2(\boldsymbol{q}_2) \times \mu_2(\boldsymbol{q}_2)).$$

For the fresh counter x_{q_2}, we can use any counter x_q that is not used in $\mu_2(\boldsymbol{q}_2)$ for other purposes. Moreover, this counter should belong to a state q with the same active counter as q_2. If multiple states q_2, q_2' with the same active counter $\mu_{\mathcal{A}}(q_2) = \mu_{\mathcal{A}}(q_2')$ need a fresh counter, we choose the same fresh counter $x_{q_2} = x_{q_2'}$ in X^d. Since each state only uses a single counter, and since we have as many counters as states, there must always be a free counter available if $\mu_2(q_2)$ needs a fresh one. Finally the edge $((\boldsymbol{q}_1, \mu_1, \succ_1), a, C^d, Y^d, (\boldsymbol{q}_2, \mu_2, \succ_2))$ is added to the transition relation $\rightarrow^d$, where $Y^d = \{x_{q_2}, \overline{x}_{q_2} \mid q_2 \in \boldsymbol{q}_2, \mu_2(q_2) = x_{q_2}\}$ is the set of fresh counters that are used in the new state.

Example 2. We apply the powerset construction to the automaton from Section 3. The resulting deterministic automaton is depicted on the next page. It has twelve reachable states $Q^d = \{Q_i, Q_i' \mid 1 \leq i \leq 6\}$ of which seven are given in the figure. The states are $Q_1 = (\{q_1\}, \{q_1 \mapsto x_1\}, \emptyset), Q_2 = (\{q_1, q_2\}, \{q_1 \mapsto x_1, q_2 \mapsto x_2\}, x_1 \succ x_2), Q_3 = (\{q_1, q_2\}, \{q_1 \mapsto x_1, q_2 \mapsto x_2\}, x_2 \succ x_1), Q_4 = (\{q_1, q_3\}, \{q_1 \mapsto x_1, q_3 \mapsto x_2\}, x_1 \succ x_2), Q_5 = (\{q_1, q_3\}, \{q_1 \mapsto x_1, q_3 \mapsto x_2\}, x_2 \succ x_1), Q_6 = (\{q_1, q_3\}, \{q_1 \mapsto x_2, q_3 \mapsto x_2\}, \emptyset)$. The states Q_i' are the states Q_i where the counters x_1 and x_2 are swapped. For space reasons we depict a check on a counter by labelling the edge with the counter. The counters x_1, x_2 observe the constraint $\{a\} \geq {}^1\!/_2$, while $\overline{x}_1, \overline{x}_2$ check the inverse constraint $\{b\} > {}^1\!/_2$.

The initial state Q_1 corresponds to the initial state q_1 in the original automaton. Under the input symbol b only the loop edge $(q_1, b, \emptyset, \emptyset, q_1)$ is enabled. Therefore for every counter constraint C^d the successor state is Q_1 again. We combined the edges $(Q_1, b, \{x_1\}, \emptyset, Q_1)$ and $(Q_1, b, \{\overline{x}_1\}, \emptyset, Q_1)$ to a single edge without counter constraints $(Q_1, b, \emptyset, \emptyset, Q_1)$. For the symbol a there are two enabled edges in the original automaton leading to q_1 and q_2. The edge to q_2 resets the counter. Therefore, we introduce a fresh counter x_2 for q_2. For computing the order between x_2 and x_1, we check the sign of x_1. The successor is state Q_2 with $x_1 \succ x_2$ if x_1 is positive and Q_3 with $x_2 \succ x_1$, otherwise.

Now we consider the outgoing edges from state Q_5. The a-labelled edges are the same as from Q_1, since there is no a-labelled edge starting from q_3. For symbol b the successor state depends on C^d. If $\overline{x}_2 \in C^d$, there are two enabled edges in the original automaton, namely $(q_1, b, \emptyset, \emptyset, q_1)$ and $(q_3, b, \emptyset, \emptyset, q_1)$. Both enter q_1 and do not reset the counter. The candidates for the new counter of q_1 are x_1 or x_2. By the linear order, x_2 is larger in Q_5, hence this is the counter used in the successor state Q_1'. If $x_2 \in C^d$, then the edge $(q_3, b, \{x_{\{a\}}\}, \emptyset, q_3)$ of the original automaton is enabled. The successor state Q_6 contains q_1 and q_3.


b
Q_1
a, x_1
$x_2, \overline{x}_2 := 0$
$a, \overline{x}_1$
$x_2, \overline{x}_2 := 0$
a, x_2
a, x_1, x_2
$b, \overline{x}_2$
$a, \overline{x}_1, \overline{x}_2$
$x_2, \overline{x}_2 := 0$
Q_2
Q_3
b
b
a, x_1
$x_2, \overline{x}_2 := 0$
$a, x_1, \overline{x}_2$
$x_2, \overline{x}_2 := 0$
$a, \overline{x}_2, \overline{x}_1$
$x_2, \overline{x}_2 := 0$
$a, \overline{x}_1$
$x_2, \overline{x}_2 := 0$
$a, \overline{x}_1$
$x_2, \overline{x}_2 := 0$
a, x_1
$x_2, \overline{x}_2 := 0$
b, x_2
Q_4
Q_5
b, x_2
Q_1'
$b, \overline{x}_2$
$b, \overline{x}_2$
Q_6
b, x_2

Proposition 4. $\mathcal{L}\left(det(\mathcal{A})\right) = \mathcal{L}\left(\mathcal{A}\right)$.

The proof is given in the appendix. A deterministic automaton is complemented by inverting the set of final states, $\overline{det(\mathcal{A})} = (Q^d, Q_0^d, Q^d \setminus Q_F^d, X^d \cup \overline{X}^d, \rightarrow^d, c^d)$.

Proposition 5. $\mathcal{L}\left(\overline{det(\mathcal{A})}\right) = \overline{\mathcal{L}\left(det(\mathcal{A})\right)}$.

The construction shows that the set of languages accepted by simple availability automata is a subset of the languages accepted by deterministic availability automata, which in turn is a subset of the languages of all availability automata. As simple automata can be simulated by one-counter machines, their emptiness problem is decidable. The intersection problem is not, since it is not decidable for flat raes. Hence, simple automata are not closed under intersection, and thus not closed under complementation. General availability automata are also not closed under complementation. Like for timed automata [AD94] one can argue that the complement of $(a + b + c)^*.c.((a + b + c)^*)_{a=1/2}.c.(a + b + c)^*$ is not accepted by any availability automaton. Since deterministic automata are closed under complementation this shows that their expressiveness is strictly between simple and general automata.

4.1 Application to Verification

Equipped with the previous complementation algorithm, we are able to tackle the following verification problems for discrete systems under availability constraints.

$$(\textbf{Syn}) \quad ? \models \mathbb{B}_{rae} \qquad\qquad (\textbf{MC}) \quad \mathcal{A} \models \mathbb{B}_{rae}.$$

The *synthesis* problem **Syn** asks for the most general availability automaton that satisfies a Boolean combination of availability expressions $\mathbb{B}_{rae}$. As usual, satisfaction is defined in terms of language inclusion. The model checking problem **MC** takes an availability automaton $\mathcal{A}$ modelling the system of interest and an availability expression $\mathbb{B}_{rae}$ that formalises the correctness condition. It reduces the problem whether $\mathcal{A}$ is a model of $\mathbb{B}_{rae}$ to a reachability query.

Our complementation algorithm is restricted to *simple* availability automata. Relying on the negation normal form for $\mathbb{B}_{rae}$, the following theorem is an immediate consequence of our previous efforts.

Theorem 3. *Consider a Boolean combination of flat raes. There is an algorithm that solves **Syn**. **MC** is reducible to reachability in availability automata.*

By Theorem 1, reachability in availability automata is of course undecidable. The definition of runs however suggests to view availability automata as particular counter automata. Therefore, Theorem 3 allows us to use state-of-the-art tools like SLAM [BR02] or BLAST [BHJM07] to solve **MC**. They support abstraction-aided reachability analysis and often successfully tackle this undecidable problem in practically relevant cases.

5 Conclusion

We defined an extension of regular expressions to specify the availability of systems. We developed a corresponding automaton model and established a full Kleene theorem. It shows that the two denote the same formal languages. An undecidability proof places the models between finite automata and counter automata. Finally, we give a complementation algorithm for the restricted simple availability automata. It yields a fully automated synthesis procedure for a practically significant class of regular availability expressions. Moreover, it allows for a reduction of availability model checking to reachability analysis.

Related work. Various extensions of regular expressions have been proposed. For example, in [ACM02] timed regular expressions are introduced and proven equivalent to timed automata of [AD94] by a timed analogue of Kleene's theorem. The availability formula in the introduction can be stated directly in real-time logics like the Duration Calculus [CH04] or investigated operationally in suitable subclasses of hybrid automata like stopwatch automata [CL00] or priced timed automata [LR08]. The advantage of the discrete setting we chose is that the essentials of availability can be studied in isolation without being overwhelmed with the technicalities of continuous timed and hybrid systems.

Availability automata are also closely related to the work on weighted automata [DKV09]. There, the alphabet is equipped with a weight function that assigns each letter a weight in some semiring. Examples include the interesting semiring $([0, 1], max, \cdot, 0, 1)$. It can be used to determine the *reliability* of a word by assigning a probability $k \in [0, 1]$ to each letter. Crucially different from our model, weighted automata do not have guards and thus the measurement does not influence the system behaviour. We employ checks and resets on the availability to mimic loop invariants as they are standard in programming languages.

In Presburger regular expressions [SSM03], a regular language is constrained by additional Presburger formulae. Our initial example of a network with 99 % availability can be specified in this formalism as $(up + down)^* \wedge x_{up} \geq 99x_{down}$. Again, as opposed to our approach, conditional executions based on intermediary valuations are not supported. Moreover, our resets allow for an unbounded number of measurements whereas Presburger regular expressions only have a finite number of arithmetic constraints.

Future work. Our results are only a first step in the study of quantitative system properties. We plan to extend the work to ω-regular and to timed languages. Also logical characterisations of availability languages seem interesting.

Practically, we envision the following application of the presented technique. By stochastic techniques, we establish availability constraints $rae_{A \gtrsim k}$ that model the components of a distributed system [dSeSG89, RS93]. When reasoning about their interaction, we abstract away the stochastic information and rely on our new availability models. Proximity to integer programs allows us to reuse efficient software model checkers for their analysis [BR02, BHJM07].

Our undecidability proof of the intersection problem (Theorem 1) requires two rae. We leave open decidability of the *emptiness problem* for a single rae.

References

[ACM02] Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. Journal of the ACM 49, 172–206 (2002)

[AD94] Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)

[BHJM07] Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker BLAST. STTT 9(5-6), 505–525 (2007)

[BR02] Ball, T., Rajamani, S.K.: The SLAM project: debugging system software via static analysis. In: Proc. of POPL, pp. 1–3. ACM, New York (2002)

[CH04] Chaochen, Z., Hansen, M.R.: Duration Calculus: A Formal Approach to Real-Time Systems. EATCS Monographs. Springer, Heidelberg (2004)

[CL00] Cassez, F., Larsen, K.G.: The impressive power of stopwatches. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 138–152. Springer, Heidelberg (2000)

[DKV09] Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata. EATCS Monographs. Springer, Heidelberg (2009)

[dSeSG89] de Souza e Silva, E., Gail, H.R.: Calculating availability and performability measures of repairable computer systems using randomization. Journal of the ACM 36(1), 171–193 (1989)

[LR08] Larsen, K.G., Rasmussen, J.I.: Automata-theoretic techniques for modal logics of programs. Theoretical Computer Science 390, 197–213 (2008)

[Min67] Minsky, M.: Computation: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs (1967)

[RS59] Rabin, M.O., Scott, D.S.: Finite automata and their decision problems. IBM Journal of Research 3(2), 115–125 (1959)

[RS93] Rubino, G., Sericola, B.: Interval availability distribution computation. In: Proc. of FTCS, pp. 48–55. IEEE, Los Alamitos (1993)

[SSM03] Seidl, H., Schwentick, T., Muscholl, A.: Numerical document queries. In: Proc. of PODS, pp. 155–166. ACM, New York (2003)

[Tri01] Trivedi, K.S.: Probability and Statistics with Reliability, Queuing, and Computer Science Applications, 2nd edn. Wiley, Chichester (2001)

Reversing Higher-Order Pi

Ivan Lanese[1], Claudio Antares Mezzina[2], and Jean-Bernard Stefani[2]

[1] Focus Team, University of Bologna/INRIA, Italy
[2] INRIA Grenoble-Rhône-Alpes, France

Abstract. The notion of reversible computation is attracting increasing interest because of its applications in diverse fields, in particular the study of programming abstractions for reliable systems. In this paper, we continue the study undertaken by Danos and Krivine on reversible CCS by defining a reversible higher-order π-calculus (HOπ). We prove that reversibility in our calculus is causally consistent and that one can encode faithfully reversible HOπ into a variant of HOπ.

1 Introduction

Motivation and contributions. The notion of reversible computation already has a long history [2]. Nowadays, it is attracting increasing interest because of its applications in diverse fields, including hardware design, biological modelling, program debugging and testing, and quantum computing. Of particular interest is its application to the study of programming abstractions for reliable systems. The work of Danos and Krivine on reversible CCS (RCCS) [5,6] provides a good example: they show how notions of reversible and irreversible actions in a process calculus can model a primitive form of transaction, an abstraction that has been found useful, in different guises, in building reliable distributed systems, including database and workflow management systems.

In this paper, we continue the study undertaken by Danos and Krivine by tackling two questions which were already anticipated in the conclusion of [5]: (i) how can we introduce reversible actions in a higher-order concurrent calculus, specifically, the asynchronous higher-order π-calculus (HOπ)? (ii) does the introduction of reversible actions augment the expressive power of HOπ? The first question finds its motivation in the pursuit of a suitable programming model for the construction of reliable and adaptive systems, and hence in the need to study the combination of reliable programming abstractions with modular dynamicity constructs enabling dynamic software update and on-line reconfiguration. The second question is motivated by language design issues: understanding which primitives bring expressive power and which do not, allows a language designer to decide which primitives to keep in a core language definition, and which to provide as derived abstractions or as part of a language library.

In response to the first question, we define a reversible variant of the higher-order π-calculus (HOπ) [11]. A general method for reversing process calculi has been proposed by Phillips and Ulidowski in [10]. Unfortunately, it is only given

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 478–493, 2010.

for calculi whose operational semantics can be defined using SOS rules conforming to the *path* format, which is not the case for HOπ [9]. We therefore adopt an approach inspired by that of Danos and Krivine, but with significant differences. In particular, in their RCCS approach, the usual congruence laws associated with the parallel operator do not hold. Our first contribution is thus a simple syntax and reduction semantics for a reversible HOπ calculus called $\rho\pi$, with a novel way to define reversible actions while preserving the usual structural congruence laws of HOπ, notably the associativity and commutativity of the parallel operator.

Concerning the second question, we settle it in the case of $\rho\pi$ by showing that, surprisingly enough, reversibility can be obtained essentially as syntactic sugar on top of HOπ. More precisely, our second contribution is a faithful compositional encoding of the $\rho\pi$ calculus into a variant of HOπ.

Outline. The paper is organized as follows. Section 2 defines the $\rho\pi$ calculus. We explain the main constructions of the calculus and we contrast our way of handling reversibility with that of Danos and Krivine. Section 3 is devoted to the proof of our first main result, namely that reverse or *backward* computations in $\rho\pi$ are causally consistent, i.e. that they proceed through configurations that are causally equivalent with configurations arising from normal or *forward* computations. Section 4 presents a compositional encoding of the $\rho\pi$ calculus into a variant of HOπ. We prove that the translation is faithful, i.e. that a $\rho\pi$ process and its encoding are weakly barbed bisimilar. Section 5 concludes the paper with a discussion of related work.

2 The $\rho\pi$ Calculus

2.1 Informal Presentation

Building a reversible variant of a process calculus involves devising appropriate syntactic representations for computation histories. In general, since a process calculus is not confluent and processes are non-deterministic, reversing a (forward) computation history means undoing the history not in a deterministic way but in a *causally consistent* fashion, where states that are reached during a backward computation are states that could have been reached during the computation history by just performing independent actions in a different order. In RCCS, Danos and Krivine achieve this with CCS without recursion by attaching a memory m to each process P, in the monitored process construct $m : P$. A memory in RCCS is a stack of information needed for processes to backtrack. Thus, if two processes P_1 and P_2 can synchronize on a channel a to evolve into P_1' and P_2', respectively, then the parallel composition of monitored processes $m_1 : (P_1 + Q_1)$ and $m_2 : (P_2 + Q_2)$ can evolve as follows:

$$m_1 : (P_1 + Q_1) \mid m_2 : (P_2 + Q_2) \rightarrow \langle m_2, a, Q_1 \rangle \cdot m_1 : P_1' \mid \langle m_1, a, Q_2 \rangle \cdot m_2 : P_2'$$

Additionally, Danos and Krivine rely on the following rule:

$$m : (P \mid Q) \equiv \langle 1 \rangle \cdot m : P \mid \langle 2 \rangle \cdot m : Q$$

so as to ensure that each primitive thread, i.e. some process of the form $R_1 + R_2$, gets its own unique identity. Unfortunately, this rule is not compatible with the usual structural congruence rules for the parallel operator, namely associativity, commutativity, and **0** as neutral element. Danos and Krivine suggest that it could be possible to work up to tree isomorphisms on memories, but this would indeed lead to a more complex syntax, as well as additional difficulties (see Remark 5 below).

We adopt for $\rho\pi$ a different approach: instead of associating each thread with a stack that records, essentially, past actions and positions in parallel branches, we rely on simple thread tags, which act as unique identifiers but have little structure, and on new process terms, which we call *memories*, which are dedicated to undoing a single (forward) computation step.

More precisely, a *forward* computation step in $\rho\pi$ (noted with arrow $\twoheadrightarrow$) consists in the receipt of a message ($\rho\pi$ is an asynchronous calculus). The receipt of a message $a\langle P\rangle$ on channel a by a receiver process (or *trigger*) $a(X) \triangleright Q$ takes in $\rho\pi$ the following form:

$$(\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q) \;\twoheadrightarrow\; \nu k.\, k : Q\{^P/_X\} \mid [M; k]$$

Each thread (message and trigger) participating in the above computation step is uniquely identified by a tag: κ_1 identifies the message $a\langle P\rangle$, and κ_2 identifies the trigger $a(X) \triangleright Q$. The result of the message receipt consists in a classical part and two side effects. The classical part is the launch of an instance $Q\{^P/_X\}$ of the body of the trigger Q, with the formal parameter X instantiated by the received value, i.e. the process P ($\rho\pi$ is a higher-order calculus). The two side effects are: (i) the tagging of the newly created process $Q\{^P/_X\}$ by a fresh name k (operator ν is the standard restriction operator of the π-calculus), and (ii) the creation of a memory process $[M; k]$. M is simply the configuration on the left hand side of the reduction, namely $M = (\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q)$.

In this setting, a *backward* computation step takes the form of an interaction between a memory and a process tagged with the appropriate name: when a memory $[M; k]$ is put in presence of a process tagged with k, a *backward* reduction (noted with the arrow $\rightsquigarrow$) can take place which kills the process tagged with k and reinstates the configuration M:

$$(k : P) \;\mid\; [M; k] \;\rightsquigarrow\; M$$

We thus have:

$$M \;\twoheadrightarrow\; \nu k.\, k : Q\{^P/_X\} \mid [M; k] \;\rightsquigarrow\; \nu k.\, M$$

Since k is fresh, $\nu k.\, M$ is actually structurally equivalent to M. We thus have a perfect reversal of a forward computation: $M \twoheadrightarrow\rightsquigarrow M$.

Remark 1. Following Danos and Krivine [6], one could consider also taking into account *irreversible* actions. We do not do so in this paper for the sake of simplicity. Adding irreversible actions to $\rho\pi$ would be conceptually straightforward.

Remark 2. Using memories as presented here to enable reversibility simplifies the formal development but leads to a space explosion of computations in $\rho\pi$. We do not consider implementation and related space efficiency issues in this paper.

$$\begin{aligned}
P, Q &::= \mathbf{0} \mid X \mid \nu a. P \mid (P \mid Q) \mid a\langle P \rangle \mid a(X) \triangleright P \\
M, N &::= \mathbf{0} \mid \nu u. M \mid (M \mid N) \mid \kappa : P \mid [\mu; k] \\
\kappa &::= k \mid \langle h, \tilde{h} \rangle \cdot k \\
\mu &::= ((\kappa_1 : a\langle P \rangle) \mid (\kappa_2 : a(X) \triangleright Q)) \\
& u \in \mathcal{I} \quad a \in \mathcal{N} \quad X \in \mathcal{V} \quad h, k \in \mathcal{K} \quad \kappa \in \mathcal{T}
\end{aligned}$$

Fig. 1. Syntax of $\rho\pi$

2.2 Syntax

Names, keys, and variables. We assume the existence of the following denumerable infinite mutually disjoint sets: the set $\mathcal{N}$ of *names*, the set $\mathcal{K}$ of *keys*, and the set $\mathcal{V}$ of process variables. The set $\mathcal{I} = \mathcal{N} \cup \mathcal{K}$ is called the set of *identifiers*. We note $\mathbb{N}$ the set of natural integers. We let (together with their decorated variants): a, b, c range over $\mathcal{N}$; h, k, l range over $\mathcal{K}$; u, v, w range over $\mathcal{I}$; X, Y, Z range over $\mathcal{V}$. We note $\tilde{u}$ a finite set of identifiers $\{u_1, \ldots, u_n\}$.

Syntax. The syntax of the $\rho\pi$ calculus is given in Figure 1 (in writing $\rho\pi$ terms, we freely add balanced parenthesis around terms to disambiguate them). *Processes* of the $\rho\pi$ calculus, given by the P, Q productions in Figure 1, are the standard processes of the asynchronous higher-order π-calculus. A receiver process (or *trigger*) in $\rho\pi$ takes the form $a(X) \triangleright P$, which allows the receipt of a message of the form $a\langle Q \rangle$ on channel a.

Processes in $\rho\pi$ cannot directly execute, only *configurations* can. *Configurations* in $\rho\pi$ are given by the M, N productions in Figure 1. A configuration is built up from *threads* and *memories*.

A *thread* $\kappa : P$ is just a tagged process P, where the tag κ is either a single key k or a pair of the form $\langle h, \tilde{h} \rangle \cdot k$, where $\tilde{h}$ is a set of keys, with $h \in \tilde{h}$. A tag serves as an identifier for a process. As we will see below, together with memories tags help capture the flow of causality in a computation.

A *memory* is a process of the form $[\mu; k]$, which keeps track of the fact that a configuration M was reached during execution, that triggered the launch of a thread tagged with the fresh tag k. In a memory $[\mu; k]$, we call μ the *configuration part* of the memory, and k the *thread tag* of the memory. Memories are generated by computation steps and are used to reverse them. The configuration part $\mu = (\kappa_1 : a\langle P \rangle) \mid (\kappa_2 : a(X) \triangleright Q)$ of the memory records the message $a\langle P \rangle$ and the trigger involved in the message receipt $a(X) \triangleright Q$, together with their respective thread tags κ_1, κ_2.

We note $\mathcal{P}$ the set of $\rho\pi$ processes, and $\mathcal{C}$ the set of $\rho\pi$ configurations. We call *agent* an element of the set $\mathcal{A} = \mathcal{P} \cup \mathcal{C}$. We let (together with their decorated variants) P, Q, R range over $\mathcal{P}$; L, M, N range over $\mathcal{C}$; and A, B, C range over agents. We call *primitive thread process*, a process that is either a message $a\langle P \rangle$ or a trigger $a(X) \triangleright P$. We let τ and its decorated variants range over primitive thread processes.

Remark 3. We have no construct for replicated processes, output prefixing, or guarded choice in $\rho\pi$: these can be easily encoded in $\rho\pi$ (in fact in asynchronous HOπ).

Free names and free variables. Notions of free identifiers and free (process) variables in $\rho\pi$ are classical. It suffices to note that constructs with binders are of the forms: $\nu a. P$ which binds the name a with scope P; $\nu u. M$, which binds the identifier u with scope M; and $a(X) \triangleright P$, which binds the variable X with scope P. We note $\mathtt{fn}(P)$, $\mathtt{fn}(M)$ and $\mathtt{fn}(\kappa)$ the set of free names, free identifiers, and free keys, respectively, of process P, of configuration M, and of tag κ. Note in particular that $\mathtt{fn}(\kappa : P) = \mathtt{fn}(\kappa) \cup \mathtt{fn}(P)$, $\mathtt{fn}(k) = \{k\}$ and $\mathtt{fn}(\langle h, \tilde{h} \rangle \cdot k) = \tilde{h} \cup \{k\}$. We say that a process P or a configuration M is closed if it has no free (process) variable. We note $\mathcal{P}^{\bullet}$ the set of closed processes, $\mathcal{C}^{\bullet}$ the set of closed configurations, and $\mathcal{A}^{\bullet}$ the set of closed agents.

Remark 4. In the remainder of the paper, we adopt *Barendregt's Variable Convention*: If terms $t_1, \ldots, t_n$ occur in a certain context (e.g. definition, proof), then in these terms all bound identifiers and variables are chosen to be different from the free ones.

Consistent configurations. Not all configurations allowed by the syntax in Figure 1 are meaningful. In a memory $[M; k]$, tags occurring in the configuration part M must be different from the thread tag k. This is because the key k is freshly generated when a computation step (a message receipt) takes place, and is used to identify the newly created thread. Tags appearing in the configuration part identify threads (message and trigger) which have participated in the computation step. In a configuration M, we require all the threads to be uniquely identified by their tag, and we require consistency between threads and memories: if M contains a memory $[N; k]$ (i.e. $[N; k]$ occurs as a subterm of M), we require M to also contain a thread tagged with k: components of this thread, i.e. threads whose tags have k as a suffix, can occur either directly in parallel with $[N; k]$ or in the configuration part of another memory contained in M (because they may have interacted with other threads). We call *consistent* a configuration that obeys these static semantic constraints. We defer the formal definition of consistent configurations to Section 2.3.

2.3 Operational Semantics

The operational semantics of the $\rho\pi$ calculus is defined via a reduction relation $\rightarrow$, which is a binary relation over closed configurations $\rightarrow \subset \mathcal{C}^{\bullet} \times \mathcal{C}^{\bullet}$, and a structural congruence relation $\equiv$, which is a binary relation over processes and configurations $\equiv \subset \mathcal{P}^2 \times \mathcal{C}^2$. We define evaluation contexts as "configurations with a hole $\cdot$" given by the following grammar:

$$\mathbb{E} ::= \cdot \;\mid\; (M \mid \mathbb{E}) \;\mid\; \nu u. \mathbb{E}$$

General contexts $\mathbb{C}$ are just processes or configurations with a hole. A congruence on processes and configurations is an equivalence relation $\mathcal{R}$ that is closed for general contexts: $P \mathcal{R} Q \implies \mathbb{C}[P] \mathcal{R} \mathbb{C}[Q]$ and $M \mathcal{R} N \implies \mathbb{C}[M] \mathcal{R} \mathbb{C}[N]$.

$$(\textsc{E.ParC})\ A \mid B \equiv B \mid A \qquad (\textsc{E.ParA})\ A \mid (B \mid C) \equiv (A \mid B) \mid C$$

$$(\textsc{E.NilM})\ A \mid \mathbf{0} \equiv A \qquad (\textsc{E.NewN})\ \nu u.\, \mathbf{0} \equiv \mathbf{0} \qquad (\textsc{E.NewC})\ \nu u.\, \nu v.\, A \equiv \nu v.\, \nu u.\, A$$

$$(\textsc{E.NewP})\ (\nu u.\, A) \mid B \equiv \nu u.\, (A \mid B) \qquad (\textsc{E.}\alpha)\ A =_\alpha B \implies A \equiv B$$

$$(\textsc{E.TagN})\ \kappa : \nu a.\, P \equiv \nu a.\, \kappa : P$$

$$(\textsc{E.TagP})\ k : \prod_{i=1}^{n} \tau_i \equiv \nu \tilde{h}.\, \prod_{i=1}^{n} (\langle h_i, \tilde{h} \rangle \cdot k : \tau_i) \quad \tilde{h} = \{h_1, \ldots, h_n\}$$

Fig. 2. Structural congruence for $\rho\pi$

The relation $\equiv$ is defined as the smallest congruence on processes and configurations that satisfies the rules in Figure 2. We note $t =_\alpha t'$ when terms t, t' are equal modulo α-conversion. If $\tilde{u} = \{u_1, \ldots, u_n\}$, then $\nu \tilde{u}.\, A$ stands for $\nu u_1. \ldots \nu u_n.\, A$. We note $\prod_{i=1}^{n} A_i$ for $A_1 \mid \ldots \mid A_n$ (there is no need to indicate how the latter expression is parenthesized because the parallel operator is associative by rule E.ParA). In rule E.TagP, processes τ_i are primitive thread processes. Recall the use of the variable convention in these rules: for instance, in the rule $(\nu u.\, A) \mid B \equiv \nu u.\, (A \mid B)$ the variable convention makes implicit the condition $u \notin \mathtt{fn}(B)$. The structural congruence rules are the usual rules for the π-calculus (E.ParC to E.α) without the rule dealing with replication, and with the addition of two new rules dealing with tags: E.TagN and E.TagP. Rule E.TagN is a scope extrusion rule to push restrictions to the top level. Rule E.TagP allows to generate unique tags for each primitive thread process in a configuration. An easy induction on the structure of terms provides us with a kind of normal form for configurations (by convention $\prod_{i \in I} A_i = \mathbf{0}$ if $I = \emptyset$):

Lemma 1 (Thread normal form). *For any configuration M, we have*

$$M \equiv \nu \tilde{u}.\, \prod_{i \in I} (\kappa_i : \rho_i) \mid \prod_{j \in J} [M_j : k_j]$$

with $\rho_i = \mathbf{0}$, $\rho_i = a_i\langle P_i \rangle$, or $\rho_i = a_i(X_i) \triangleright P_i$.

Remark 5. One could have thought of mimicking the structural congruence rule dealing with parallel composition in [5], using a monoid structure for tags:

$$(\textsc{E.TagP}^\bullet)\ \kappa : (P \mid Q) \equiv \nu h_1, h_2.\, (h_1 \cdot \kappa : P) \mid (h_2 \cdot \kappa : Q)$$

Unfortunately using E.TagP$^\bullet$ instead of E.tagP would introduce some undesired non-determinism, which would later complicate our definitions (in relation to causality) and proofs. For instance, let $M = k : a\langle Q \rangle \mid (h : a(X) \triangleright X)$. We have: $M \to M' = \nu l.\, (l : Q) \mid [M; l]$ Now, assuming E.tagP$^\bullet$, we would have

$$M \equiv (k : (a\langle Q \rangle \mid \mathbf{0})) \mid (h : a(X) \triangleright X) \equiv \nu h_1, h_2.\, ((h_1 \cdot k : a\langle Q \rangle) \mid (h_2 \cdot k : \mathbf{0})) \mid (h : a(X) \triangleright X)$$

$$(\text{R.Fw})\quad (\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q) \twoheadrightarrow \nu k.\,(k : Q\{^P/_X\}) \mid [(\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q); k]$$

$$(\text{R.Bw})\quad (k : P) \mid [M; k] \rightsquigarrow M$$

Fig. 3. Reduction rules for $\rho\pi$

Let $M_1 = (h_1 \cdot k : a\langle Q\rangle) \mid (h : a(X) \triangleright X)$. We would then have: $M \to M''$, where $M'' = \nu h_1, h_2, l.\,(l : Q) \mid [M_1; l] \mid (h_2 \cdot k : \mathbf{0})$. Clearly $M' \not\equiv M''$, which means that a seemingly deterministic configuration, M, would have in fact two (actually, an infinity of) derivations towards non structurally equivalent configurations. By insisting on tagging only primitive thread processes, E.TagP avoids this unfortunate situation.

We can characterize this by proving a kind of *determinacy* lemma for $\rho\pi$, which fails if we replace rule E.TagP with rule E.TagP$^\bullet$. Extend the grammar of $\rho\pi$ with marked primitive thread processes of the form $\tau^\bullet$. This extended calculus has exactly the same structural congruence and reduction rules than $\rho\pi$, but with possibly marked primitive thread processes. Now call *primed* a closed configuration M with exactly two marked processes of the form $a\langle P\rangle^\bullet$ and $(a(X) \triangleright Q)^\bullet$. Anticipating on the definition of the reduction relation $\to$ below (with $\equiv$ and $\to$ trivially extended to marked processes), we have:

Lemma 2 (Determinacy). *Let M be a primed configuration such that $M \equiv M_1 = \mathbb{E}_1[\kappa_1 : a\langle P\rangle^\bullet \mid \kappa_2 : (a(X) \triangleright Q)^\bullet]$ and $M \equiv M_2 = \mathbb{E}_2[\kappa_1' : a\langle P\rangle^\bullet \mid \kappa_2' : (a(X) \triangleright Q)^\bullet]$. Assume $M_1 \to M_1'$ and $M_2 \to M_2'$ are derived by applying* R.Fw *with configurations $\kappa_1 : a\langle P\rangle^\bullet \mid \kappa_2 : (a(X) \triangleright Q)^\bullet$, and $\kappa_1' : a\langle P\rangle^\bullet \mid \kappa_2' : (a(X) \triangleright Q)^\bullet$, respectively, followed by* R.Ctx. *Then $M_1' \equiv M_2'$.*

Proof. By induction on the form of $\mathbb{E}_1$, and case analysis on the form of κ_1 and κ_2. □

We say that a binary relation $\mathcal{R}$ on closed configurations is *evaluation-closed* if it satisfies the inference rules:

$$(\text{R.Ctx})\ \frac{M \mathrel{\mathcal{R}} N}{\mathbb{E}[M] \mathrel{\mathcal{R}} \mathbb{E}[N]} \qquad (\text{R.Eqv})\ \frac{M \equiv M' \qquad M' \mathrel{\mathcal{R}} N' \qquad N' \equiv N}{M \mathrel{\mathcal{R}} N}$$

The reduction relation $\to$ is defined as the union of two relations, the *forward* reduction relation $\twoheadrightarrow$ and the *backward* reduction relation $\rightsquigarrow$: $\to\ =\ \twoheadrightarrow \cup \rightsquigarrow$. Relations $\twoheadrightarrow$ and $\rightsquigarrow$ are defined to be the smallest evaluation-closed binary relations on closed configurations satisfying the rules in Figure 3 (note again the use of the variable convention: in rule R.Fw the key k is fresh).

The rule for forward reduction (R.Fw) is the standard communication rule of the higher-order π-calculus with two side effects: (i) the creation of a new memory to record the configuration that gave rise to it, namely the parallel composition of a message and a trigger, properly tagged (tags κ_1 and κ_2 in the rule); (ii) the tagging of the continuation of the message receipt (with the fresh key k). The rule for backward reduction (R.Bw) is straightforward: in presence of the thread tagged with key k, memory $[M; k]$ reinstates the configuration M that gave rise to the tagged thread.

We can now formally define the notion of *consistent configuration.*

Definition 1 (Consistent configuration). *A configuration* $M \equiv \nu\tilde{u}.\prod_{i\in I}(\kappa_i : \rho_i) \mid \prod_{j\in J}[M_j; k_j]$, *with* $\rho_i = \mathbf{0}$ *or* ρ_i *a primitive thread process,* $M_j = \delta_j : R_j \mid \gamma_j : T_j$, $R_j = a_j\langle P_j\rangle$, $T_j = a_j(X_j) \triangleright Q_j$, *is said to be* consistent *if the following conditions are met:*

1. *For all* $j \in J$, $k_j \neq \delta_j$, $k_j \neq \gamma_j$ *and* $\delta_j \neq \gamma_j$
2. *For all* $i_1, i_2 \in I$, $i_1 \neq i_2 \implies \kappa_{i_1} \neq \kappa_{i_2}$
3. *For all* $i \in I, j \in J$, $\kappa_i \neq \delta_j$ *and* $\kappa_i \neq \gamma_j$
4. *For all* $j \in J$, *there exist* $E \subseteq I$, $D \subseteq J$, $G \subseteq J$, *such that:*

$$\nu\tilde{u}.\, k_j : Q_j\{^{P_j}/_{X_j}\} \equiv \nu\tilde{u}.\prod_{e\in E}\kappa_e : \rho_e \mid \prod_{d\in D}\delta_d : R_d \mid \prod_{g\in G}\gamma_g : T_g$$

Consistent configurations are preserved by reduction:

Lemma 3. *Let M be a consistent configuration. If $M \to N$ then N is a consistent configuration.*

Proof. By case analysis on the derivation of $M_t \to N$, where $M \equiv M_t$ and M_t is in thread normal form. □

Barbed bisimulation. The operational semantics of the $\rho\pi$ calculus is completed classically by the definition of a contextual equivalence between configurations, which takes the form of a barbed congruence. We first define observables in configurations. We say that name a is *observable in configuration* M, noted $M \downarrow_a$, if $M \equiv \nu\tilde{u}.\,(\kappa : a\langle P\rangle) \mid N$, with $a \notin \tilde{u}$. Note that keys are not observable: this is because they are just an internal device used to support reversibility. We note $\Rightarrow$ the reflexive and transitive closure of $\to$. The following definitions are classical:

Definition 2 (Barbed bisimulation and congruence). *A relation* $\mathcal{R} \subseteq \mathcal{C}^\bullet \times \mathcal{C}^\bullet$ *on closed configurations is a* strong *(resp.* weak*)* barbed simulation *if whenever* $M \,\mathcal{R}\, N$

- $M \downarrow_a$ *implies* $N \downarrow_a$ *(resp.* $N \Rightarrow\downarrow_a$*)*
- $M \to M'$ *implies* $N \to N'$, *with* $M'\mathcal{R}N'$ *(resp.* $N \Rightarrow N'$ *with* $M'\mathcal{R}N'$*)*

A relation $\mathcal{R} \subseteq \mathcal{C}^\bullet \times \mathcal{C}^\bullet$ *is a* strong *(resp.* weak*)* barbed bisimulation *if* $\mathcal{R}$ *and* $\mathcal{R}^{-1}$ *are strong (resp. weak) barbed simulations. We call* strong *(resp.* weak*)* barbed bisimilarity *and note* $\dot{\sim}$ *(resp.* $\dot{\approx}$*) the largest strong (resp. weak) barbed bisimulation. The largest congruence included in* $\dot{\sim}$ *(resp.* $\dot{\approx}$*) is called* strong *(resp.* weak*)* barbed congruence *and is noted* $\sim$ *(resp.* $\approx$*).*

2.4 Basic Properties of Reduction in $\rho\pi$

Define inductively the *erasing function* $\gamma : \mathcal{C} \to \mathcal{P}$, which maps a configuration on its underlying HOπ process, by the following clauses:

$$\begin{array}{lll} \gamma(\mathbf{0}) = \mathbf{0} & \gamma(\nu a.\, M) = \nu a.\, \gamma(M) & \gamma(M \mid N) = \gamma(M) \mid \gamma(N) \\ \gamma(\kappa : P) = P & \gamma([M; k]) = \mathbf{0} & \gamma(\nu k.\, M) = M \end{array}$$

The following lemma shows that $\rho\pi$ forward computations are indeed decorations on HOπ reductions.

Lemma 4. *For all closed configurations M, N, if $M \twoheadrightarrow N$ then $\gamma(M) \to_\pi \gamma(N)$*

Proof. Straightforward, by first proving by induction on the derivation of $M \equiv N$ that $M \equiv N \implies \gamma(M) \equiv \gamma(N)$, and then by reasoning by induction on the derivation of $M \twoheadrightarrow N$. □

Remark 6. One can lift a closed HOπ process P to a closed consistent configuration in $\rho\pi$ by defining $\delta(P) = \nu k.\, k : P$.

The next lemma shows that forward and backward reductions in $\rho\pi$ are really inverse of one another.

Lemma 5 (Loop lemma). *For all closed consistent configurations M, N, $M \twoheadrightarrow N \iff N \rightsquigarrow M$.*

Proof. By induction on the derivation of $M \twoheadrightarrow N$ for the *if* direction, and on the derivation of $N \rightsquigarrow M$ for the converse. □

A direct consequence of the Loop Lemma is that a closed consistent configuration M is weakly barbed congruent to any of its descendants, a fact that one can understand as a kind of observational property of reversibility:

Lemma 6 (Observational equivalence). *If $M \Rightarrow N$, then $M \approx N$.*

Proof. By induction on the form of configuration contexts, the base case $M \stackrel{\cdot}{\approx} N$ being obtained by applying the Loop Lemma. □

3 Causality in $\rho\pi$

We now proceed to the analysis of causality in $\rho\pi$, showing that reversibility in $\rho\pi$ is causally consistent. We mostly adapt for the exposition the terminology and arguments of [5].

We call *transition* a triplet of the form $M \xrightarrow{m} M'$, or $M \xrightarrow{m_\bullet} M'$, where M, M' are closed consistent configurations, $M \to M'$, and m is the memory involved in the reduction $M \to M'$. We say that a memory m is *involved* in a reduction $M \twoheadrightarrow M'$ if $M \equiv \mathbb{E}[\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q]$, $M' \equiv \mathbb{E}[\nu k.\, (k : Q\{^P/_X\}) \mid m]$, and $m = [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k]$. In this case, the transition involving m is noted $M \xrightarrow{m} M'$. Likewise, we say that a memory $m = [N; k]$ is involved in a reduction $M \rightsquigarrow M'$ if $M \equiv \mathbb{E}[(k : Q) \mid m]$, $M' \equiv \mathbb{E}[N]$. In this case, the transition involving m is noted $M \xrightarrow{m_\bullet} M'$.

We say a transition $t : M \xrightarrow{m} M'$ is *name-preserving* if M and M' are in thread normal form and if either (i) $M = \nu\tilde{u}.\, \prod_{i \in I}(\kappa_i : \rho_i) \mid \prod_{j \in J}[M_j : k_j]$,

$M' = \nu\tilde{u}.\ \prod_{i\in I'}(\kappa_i : \rho_i) \mid \prod_{j\in J'}[M_j : k_j]$, with $I' \subseteq I, J \subseteq J'$, and $m = [N_j; k_j]$ for some $j \in J'$; or (ii) $M = \nu\tilde{u}.\ \prod_{i\in I}(\kappa_i : \rho_i) \mid \prod_{j\in J}[M_j : k_j]$, $M' = \nu\tilde{u}.\ \prod_{i\in I'}(\kappa_i : \rho_i) \mid \prod_{j\in J'}[M_j : k_j]$, with $I \subseteq I'$, $J' \subseteq J$, and $m = [M_j; k_j]$ for some $j \in J$. Intuitively, a name-preserving transition keeps track of the set of restricted names of its configurations, all the names used in the transition (and especially the tag of memory m). In the rest of this section we only consider name-preserving transitions and "transition" used in a definition, lemma or theorem, stands for "name-preserving transition". Note that working with name-preserving transitions only is licit because of the determinacy lemma (Lemma 5).

In a transition $M \xrightarrow{\mu} N$, we say that M is the *source* or the transition, N is its *target*, and μ is its label (of the form m or $m_\bullet$, where m is some memory – we let μ and its decorated variants range over transition labels). If $\mu = m_\bullet$, we set $\mu_\bullet = m$. Two transitions are said to be *coinitial* if they have the same source, *cofinal* if they have the same target, *composable* if the target of one is the source of the other. A sequence of pairwise composable transitions is called a *trace*. We let t and its decorated variants range over transitions, σ and its decorated variants range over traces. Notions of target, source and composability extend naturally to traces. We note ϵ_M the empty trace with source M, $\sigma_1; \sigma_2$ the composition of two composable traces σ_1 and σ_2. The *stamp* $\lambda(m)$ of a memory $m = [\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k]$ is defined to be the set $\{\kappa_1, \kappa_2, k\}$; we set $\lambda(m_\bullet) = \lambda(m)$.

Definition 3 (Concurrent transitions). *Two coinitial transitions $t_1 = M \xrightarrow{\mu_1} M_1$ and $t_2 = M \xrightarrow{\mu_2} M_2$ are said to be* concurrent *if $\lambda(\mu_1) \cap \lambda(\mu_2) = \emptyset$.*

Remark 7. Note that the stamp of a memory $[M; k]$ include its tag k. This is necessary to take into account possible conflicts between a forward action and a backward action.

The Loop Lemma ensures that each transition $t = M \xrightarrow{\mu} N$ has a reverse one $t_\bullet = N \xrightarrow{\mu_\bullet} M$. The above definition of concurrent transitions makes sense:

Lemma 7 (Square lemma). *If $t_1 = M \xrightarrow{\mu_1} M_1$ and $t_2 = M \xrightarrow{\mu_2} M_2$ are two coinitial concurrent transitions, then there exist two cofinal transitions $t_2/t_1 = M_1 \xrightarrow{\mu_2} N$ and $t_1/t_2 = M_2 \xrightarrow{\mu_1} N$.*

Proof. By case analysis on the form of transitions t_1 and t_2. □

We are now in a position to show that reversibility in $\rho\pi$ is causally consistent. We define first the notion of causal equivalence between traces, noted $\asymp$, as the least equivalence relation between traces closed under composition that obeys the following rules:

$$t_1; t_2/t_1 \asymp t_2; t_1/t_2 \qquad t; t_\bullet \asymp \epsilon_{\mathtt{source}(t)} \qquad t_\bullet; t \asymp \epsilon_{\mathtt{target}(t)}$$

The proof of causal consistency proceeds along the exact same lines as in [5], with simpler arguments because of the simpler form of our memory stamps.

Lemma 8 (Rearranging Lemma). *Let σ be a trace. There exists forward traces σ' and σ'' such that $\sigma \asymp \sigma'_\bullet; \sigma''$.*

$$\begin{aligned}
P, Q &::= \mathbf{0} \mid X \mid \nu u.\, P \mid (P \mid Q) \mid u\langle F, v\rangle \mid J \triangleright P \mid (F\ V)\\
F &::= (u)P \mid (X)P\\
V &::= u \mid F\\
J &::= u(X, v) \mid u(X, \backslash v) \mid J \mid J\\
& u, v \in \mathcal{I}
\end{aligned}$$

Fig. 4. Syntax of HOπ^+

Lemma 9 (Shortening Lemma). *Let σ_1, σ_2 be coinitial and cofinal traces, with σ_2 forward. Then, there exists a forward trace σ'_1 of length at most that of σ_1 such that $\sigma'_1 \asymp \sigma_1$.*

Theorem 1 (Causal consistency). *Let σ_1 and σ_2 be coinitial traces, then $\sigma_1 \asymp \sigma_2$ if and only if σ_1 and σ_2 are cofinal.*

4 Encoding $\rho\pi$ in HOπ^+

We show in this section that $\rho\pi$ can be encoded in a variant of HOπ, with bi-adic channels, join patterns, sub-addressing, abstractions and applications, which we call HOπ^+. This particular variant was chosen for convenience, because it simplifies our encoding. All HOπ^+ constructs are well understood in terms of expressive power with respect to HOπ and π [12,8,4]. The syntax of HOπ^+ is given in Figure 4. Channels in HOπ^+ carry both a name and an abstraction (bi-adicity); a trigger can receive a message on a given channel, or on a given channel provided the received message carries some given name (sub-addressing). HOπ^+ has abstractions over names $(u)P$ and over process variables $(X)P$, and applications $(P\ V)$, where a value V can be a name or an abstraction. We take the set of names of HOπ^+ to be the set $\mathcal{I} \cup \{\star\}$ of $\rho\pi$. The set of (process) variables of HOπ^+ is taken to coincide with the set $\mathcal{V}$ of variables of $\rho\pi$.

The structural congruence for HOπ^+, also noted $\equiv$, obeys the same rules than those of $\rho\pi$, minus the rules E.TagN and E.TagP, which are specific to $\rho\pi$. Evaluation contexts in HOπ^+ are are given by the following grammar:

$$\mathbb{E} ::= \cdot \mid (P \mid \mathbb{E}) \mid \nu u.\, \mathbb{E}$$

The reduction relation for HOπ^+, also noted $\rightarrow$, is defined as the least evaluation closed relation (same definition as for $\rho\pi$, with HOπ^+ processes instead of configurations) that satisfies the rules in Figure 5. The function `match` in Figure 5 is the partial function which is defined in the cases given by the clauses below, and undefined otherwise:

$$\mathtt{match}(u, v) = \{^{u}/_{v}\} \quad \mathtt{match}(u, \backslash u) = \{^{u}/_{u}\} \quad \mathtt{match}(F, X) = \{^{F}/_{X}\}$$

$$(\textsc{Red})\ \prod_{i=1}^{n} u_i\langle F_i, v_i\rangle \mid (\prod_{i=1}^{n} u_i(X_i,\psi_i) \triangleright P) \to P\{^{F_1\dots F_n}/_{X_1\dots X_n}\}\theta_1\dots\theta_n \qquad \mathtt{match}(v_i,\psi_i)=\theta_i$$

$$(\textsc{App})\ ((\psi)F\ V) \to F\theta \qquad \mathtt{match}(V,\psi)=\theta$$

Fig. 5. Reduction rules for HOπ^+

$$(\!|\mathbf{0}|\!) = \mathtt{Nil} \qquad\qquad (\!|X|\!) = (l)(X\ l)$$

$$(\!|a\langle P\rangle|\!) = (l)(\mathtt{Msg}\ a\ (\!|P|\!)\ l) \qquad\qquad (\!|\nu a.\, P|\!) = (l)\nu a.\, (\!|P|\!)\, l$$

$$(\!|P \mid Q|\!) = (l)(\mathtt{Par}\ (\!|P|\!)\ (\!|Q|\!)\ l) \qquad\qquad (\!|a(X) \triangleright P|\!) = (l)(\mathtt{Trig}_{(\!|P|\!)}\ a\ l)$$

$$\mathtt{Nil} = (l)l\langle\mathtt{Nil}\rangle$$

$$\mathtt{Msg} = (a\ X\ l)a\langle X, l\rangle \mid (\mathtt{KillM}\ a\ X\ l)$$

$$\mathtt{KillM} = (a\ X\ l)(a(X, \backslash l) \triangleright l\langle(h)\mathtt{Msg}\, a\, X\, h\rangle \mid \mathtt{Rew}\, l)$$

$$\mathtt{Par} = (X\ Y\ l)\nu h, k.\, X\, h \mid Y\, k \mid (\mathtt{KillP}\ h\ k\ l)$$

$$\mathtt{KillP} = (h\ k\ l)(h(W) \mid k(Z) \triangleright l\langle(l)\mathtt{Par}\, W\, Z\, l\rangle \mid \mathtt{Rew}\, l)$$

$$\mathtt{Trig}_{(\!|P|\!)} = (a\ l)\nu\mathtt{t}.\, \mathtt{t} \mid (a(X,h) \mid \mathtt{t} \triangleright \nu k.\, (\!|P|\!)\, k \mid (\mathtt{Mem}_{(\!|P|\!)}\ a\ X\ h\ k\ l)) \mid (\mathtt{KillT}_{(\!|P|\!)}\ \mathtt{t}\ l\ a)$$

$$\mathtt{KillT}_{(\!|P|\!)} = (\mathtt{t}\ l\ a)(\mathtt{t} \triangleright l\langle(h)\mathtt{Trig}_{(\!|P|\!)}\, a\, h\rangle \mid \mathtt{Rew}\, l)$$

$$\mathtt{Mem}_{(\!|P|\!)} = (a\ X\ h\ k\ l)k(Z) \triangleright (\mathtt{Msg}\ a\ X\ h) \mid (\mathtt{Trig}_{(\!|P|\!)}\ a\ l)$$

$$\mathtt{Rew} = (l)(l(Z) \triangleright Z\, l)$$

Fig. 6. Encoding $\rho\pi$ processes

Conventions. In writing HOπ^+ terms, $u\langle v\rangle$ abbreviates $u\langle(X)\mathbf{0}, v\rangle$ and $u\langle F\rangle$ abbreviates $u\langle F, \star\rangle$. Likewise, $a(u)\triangleright P$ abbreviates $a(X,u)\triangleright P$, where $X \not\in \mathtt{fv}(P)$, and $u(X)\triangleright P$ abbreviates $a(X,\star)\triangleright P$. We adopt the usual conventions for writing applications and abstractions: $(F\ V_1\dots V_n)$ stands for $(((F\ V_1)\dots)\ V_n)$, and $(X_1\dots X_n)F$ stands for $(X_1)\dots(X_n)F$. When there is no potential ambiguity, we often write $F\,V$ for $(F\ V)$. When defining HOπ^+ processes, we freely use recursive definitions for these can be encoded using e.g. the Turing fixed point combinator Θ defined as $\Theta = (\mathtt{A}\ \mathtt{A})$, where $\mathtt{A} = (X\ F)(F\ (X\ X\ F))$ (cf. [1] p.132).

The encoding $(\!|\cdot|\!) : \mathcal{P}_{\rho\pi} \to \mathcal{P}_{\mathrm{HO}\pi^+}$ of processes of $\rho\pi$ in HOπ^+ is defined inductively in Figure 6. It extends to an encoding $(\!|\cdot|\!) : \mathcal{C}_{\rho\pi} \to \mathcal{P}_{\mathrm{HO}\pi^+}$ of configurations of $\rho\pi$ in HOπ^+ as given in Figure 7 (note that the encoding for $\mathbf{0}$ in Figure 7 is the encoding for the null configuration). The main idea behind the encoding is simple: a tagged process $l : P$ is interpreted as a process equipped with a special channel l on which to report that it has successfully rolled back. This intuition leads to the encoding of a $\rho\pi$ process as an abstraction which takes this reporting channel l as a parameter. Rolling back a message is simply consuming it and sending it (actually the message encoding itself) on the report channel. Rolling back a trigger is just consuming the special token $\mathtt{t}$ that locks it and sending the trigger encoding on the report channel. Rolling back a parallel composition is just rolling back its branches and reporting when all have rolled back. The last

$$
\begin{aligned}
&(\!|\mathbf{0}|\!) = \mathbf{0}\\
&(\!|M \mid N|\!) = (\!|M|\!) \mid (\!|N|\!)\\
&(\!|\nu u.\, M|\!) = \nu u.\, (\!|M|\!)\\
&(\!|k : P|\!) = ((\!|P|\!)\ k)\\
&(\!|\langle h_i, \tilde{h}\rangle \cdot k : P|\!) = ((\!|P|\!)\ h_i) \mid \mathtt{Kill}_{\langle h_i,\tilde{h}\rangle\cdot k}\\
&(\!|[\kappa_1 : a\langle P\rangle \mid \kappa_2 : a(X) \triangleright Q; k]|\!) = (\mathtt{Mem}_{(\!|Q|\!)}\ a\ (\!|P|\!)\ (\!|\kappa_1|\!)\ k\ (\!|\kappa_2|\!)) \mid \mathtt{Kill}_{\kappa_1} \mid \mathtt{Kill}_{\kappa_2}\\
&(\!|k|\!) = k\\
&(\!|\langle h_i, \tilde{h}\rangle \cdot k|\!) = h_i
\end{aligned}
$$

$$
\begin{aligned}
&\mathtt{Kill}_{\langle h_1,\tilde{h}\rangle\cdot k} = \nu\tilde{l}.\, (\mathtt{KillP}\ h_1\ h_2\ l_1) \mid \ldots \mid (\mathtt{KillP}\ h_{n-1}\ l_{n-3}\ l_{n-2}) \mid (\mathtt{KillP}\ h_n\ l_{n-2}\ k)\\
&\mathtt{Kill}_{\kappa} = \mathbf{0} \quad \text{otherwise}
\end{aligned}
$$

Fig. 7. Encoding $\rho\pi$ configurations

part of this section is devoted to prove that the encoding is faithful, i.e. that it preserves the semantics of the original process. More precisely, we will prove the following theorem:

Theorem 2 (Operational correspondance). *For any closed $\rho\pi$ process P, $\nu k.\ k : P \dot{\approx} (\!|\nu k.\ k : P|\!)$.*

One cannot simply prove that given a (consistent) configuration M, if $M \to M'$ then $(\!|M|\!) \Rightarrow (\!|M'|\!)$. In fact this does not hold, since the translated processes produce some garbage, and since structural congruent processes do not always have structural congruent translations. Thus we need some auxiliary machinery.

Definition 4. *Let $\equiv_{Ex}$ be the smallest congruence satisfying the rules for structural congruence $\equiv$ plus the rules below.*

$$(\text{Ex.Nil})\ \nu l_1, l_2.\, l_1\langle \mathtt{Nil}\rangle \mid ((\!|R|\!) l_2) \mid (\mathtt{KillP}\ l_1\ l_2\ l) \equiv_{Ex} ((\!|R|\!) l)$$

$$(\text{Ex.A})\ \nu l'.\, (\mathtt{KillP}\ l_1\ l_2\ l') \mid (\mathtt{KillP}\ l'\ l_3\ l) \equiv_{Ex} \nu l'.\, (\mathtt{KillP}\ l_1\ l'\ l) \mid (\mathtt{KillP}\ l_2\ l_3\ l')$$

$$(\text{Ex.Unfold})\ (\!|P|\!) l \equiv_{Ex} l\langle (\!|P|\!)\rangle \mid (\mathtt{Rew}\, l)$$

Furthermore let $\equiv_C$ be the smallest congruence satisfying rules for structural congruence $\equiv$ plus the first two rules above.

Definition 5. *Let P be a HOπ^+ process. Then $\mathtt{addG}(P)$ is any process obtained from P by applying one or more times the following rules:*

$$
\begin{aligned}
&\mathbb{C}[P'] \mapsto \mathbb{C}[P' \mid (\mathtt{Rew}\, l)] && \mathbb{C}[P'] \mapsto \mathbb{C}[P' \mid (\mathtt{KillM}\ a\ X\ l)]\\
&\mathbb{C}[P'] \mapsto \mathbb{C}[P' \mid (\mathtt{KillP}\ h\ k\ l)] && \mathbb{C}[P'] \mapsto \mathbb{C}[P' \mid \nu t.\, (a(X,k) \mid t \triangleright Q)]
\end{aligned}
$$

We characterize now the effect of the encoding on structural congruent configurations.

Lemma 10. *Let M, N be configurations. Then $M \equiv N$ implies $(\!| M |\!) \equiv_C (\!| N |\!)$.*

It is easy to see that names in $\mathcal{K}$ are always bound.

Lemma 11. *If $(\!| \nu k.\, k : P |\!) \Rightarrow P'$ then $\mathtt{fn}(P') \cap \mathcal{K} = \emptyset$.*

The next lemma shows that the encoding of a process can mimic its reductions.

Lemma 12. *For each consistent configuration M, if $M \to M'$ then $(\!| M |\!) \Rightarrow P$ with $P \equiv_{Ex} \mathtt{addG}((\!| M' |\!))$.*

Proof. By induction on the derivation of $M \to M'$. The two base cases correspond to $\twoheadrightarrow$ and $\rightsquigarrow$. The garbage produced by the translation is managed by function addG. For closure under context, the proof is easy by induction on the structure of the context. Closure under structural congruence is managed by relation $\equiv_C$. □

We can now prove our main result.

Proof (of Theorem 2). We have to prove that the following relation is a barbed bisimulation:

$$\mathcal{R} = \{(M, N) \mid \nu k.\ k : P \Rightarrow M \wedge N \equiv_{Ex} \mathtt{addG}((\!| M |\!))\}$$

For names in $\mathcal{K}$, the condition on barbs follows from Lemma 11. For names in $\mathcal{N}$, it is proved by observing that $\equiv_{Ex}$ and $\mathtt{addG}(\bullet)$ do not change those barbs. For terms of the form $(\!| M |\!)$ the condition is proved by structural induction on M.

Then we have to show that each reduction of M is matched by a reduction of N and viceversa. The first direction follows from Lemma 12. It is easy to see that all the processes related to M can simulate its transitions.

For the other direction we have a case analysis according to the channels involved in the reduction $N \to N'$. Notably, since all reductions are reversible, reductions can not change the set of weak barbs. The case analysis follows:

- reductions involving a message on a name $a \in \mathit{names}$: these correspond to a transition $M \twoheadrightarrow M'$, and it is easy to see that $M' \mathcal{R} N'$;
- reductions involving a memory: these correspond to a transition $M \rightsquigarrow M'$, and it is easy to see that $M' \mathcal{R} N'$;
- reductions involving a Kill process: these do not correspond to any transition of M, and it is easy to see that $M \mathcal{R} N'$.

Note that garbage does not add further reductions. This is trivial for triggers with a bound premise. This will hold for other triggers too, since for each k there is at most one message on k at the time, and when such a message is produced the trigger consuming it is produced too. Thus additional triggers are redundant. □

5 Conclusion and Related Work

We have presented a reversible asynchronous higher-order π-calculus, which we have shown to be causally consistent. The paper gets its inspiration from Danos and Krivine work [5] and makes two original contributions. The first one is a novel way to introduce reversibility in a process calculus which preserves the classical structural congruence laws of the π-calculus, and which relies on simple name tags for identifying threads and explicit memory processes, compared to the two previous approaches of RCCS [5], that relied on memory stacks as thread tags, and of Phillips and Ulidowski[10], that relied on making the structure of terms in SOS rules static and on keeping track of causality by tagging actions in SOS rules. A further paper by Danos et al. [7] provides an abstract categorical analysis of the RCCS constructions, but it leaves intact the question of devising an appropriate "syntactic representation of the reversible history category" for the target formalism (in our case, asynchronous HOπ), which is not entirely trivial. The second contribution of the paper is a faithful encoding on our reversible HOπ calculus into a variant of HOπ, showing that adding reversibility does not change substantially the expressive power of HOπ. This result is consistent with, though not reducible to, Boreale and Sangiorgi's encoding of the π-calculus with causality in the π-calculus itself [3]. Our encoding trades simplicity and weak barbed bisimilarity for divergence in several places. It would be interesting to see whether divergence added by the encoding can be eliminated while still preserving weak bisimilarity, and furthermore whether we can extend our result to obtain full abstraction (proving that weak barbed congruence on $\rho\pi$ configurations corresponds to weak barbed congruence of their encodings).

Acknowledgments. Many thanks to Cosimo Laneve and Alan Schmitt for interesting suggestions, and to anonymous referees for suggested clarifications.

References

1. Barendregt, H.P.: The Lambda Calculus – Its Syntax and Semantics. North-Holland, Amsterdam (1984)
2. Bennett, C.H.: Notes on the history of reversible computation. IBM Journal of Research and Development 32(1) (1988)
3. Boreale, M., Sangiorgi, D.: A fully abstract semantics for causality in the π-calculus. Acta Informatica 35(5) (1998)
4. Carbone, M., Maffeis, S.: On the expressive power of polyadic synchronisation in pi-calculus. Nord. J. Comput. 10(2) (2003)
5. Danos, V., Krivine, J.: Reversible communicating systems. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 292–307. Springer, Heidelberg (2004)
6. Danos, V., Krivine, J.: Transactions in RCCS. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 398–412. Springer, Heidelberg (2005)

7. Danos, V., Krivine, J., Sobocinski, P.: General reversibility. Electr. Notes Theor. Comput. Sci. 175(3) (2007)
8. Fournet, C., Gonthier, G.: The reflexive chemical abstract machine and the join-calculus. In: 23rd ACM Symp. on Princ. of Prog. Languages, POPL (1996)
9. Mousavi, M.R., Reniers, M.A., Groote, J.F.: SOS formats and meta-theory: 20 years after. Theor. Comput. Sci. 373(3) (2007)
10. Phillips, I., Ulidowski, I.: Reversing algebraic process calculi. J. Log. Algebr. Program. 73(1-2) (2007)
11. Sangiorgi, D.: Bisimulation for higher-order process calculi. Information and Computation 131(2) (1996)
12. Sangiorgi, D., Walker, D.: The π-calculus: A Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)

Modal Logic over Higher Dimensional Automata

Cristian Prisacariu*

Dept. of Informatics, Univ. of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway
cristi@ifi.uio.no

Abstract. Higher dimensional automata (*HDAs*) are a model of concurrency that can express most of the traditional partial order models like Mazurkiewicz traces, pomsets, event structures, or Petri nets. Modal logics, interpreted over Kripke structures, are the logics for reasoning about sequential behavior and interleaved concurrency. Modal logic is a well behaved subset of first-order logic; many variants of modal logic are decidable. However, there are no modal-like logics for the more expressive *HDA* models. In this paper we introduce and investigate a modal logic over *HDAs* which incorporates two modalities for reasoning about "during" and "after". We prove that this general higher dimensional modal logic (*HDML*) is decidable and we define a complete axiomatic system for it. We also show how, when the *HDA* model is restricted to Kripke structures, a syntactic restriction of *HDML* becomes the standard modal logic. Then we isolate the class of *HDAs* that encode Mazurkiewicz traces and show how *HDML* can be restricted to LTrL (the linear time temporal logic over Mazurkiewicz traces).

1 Introduction

Higher dimensional automata (*HDAs*) are a general formalism for modeling concurrent systems [14,21]. In this formalism concurrent systems can be modeled at different levels of abstraction, not only as all possible interleavings of their concurrent actions. *HDAs* can model concurrent systems at any granularity level and make no assumptions about the durations of the actions. Moreover, *HDAs* are not constrained to only before-after modeling and expose explicitly the choices in the system. It is a known issue in concurrency models that the combination of causality, concurrency, and choice is difficult; in this respect, *HDAs* and Chu spaces [15] do a fairly good job [17].

Higher dimensional automata are more expressive than most of the models based on partial orders or on interleavings (e.g., Petri nets and the related Mazurkiewicz traces, or the more general partial order models like pomsets or event structures). Therefore, one only needs to find the right class of *HDAs* in order to get the desired models of concurrency.

* Partially supported by the Nordunet3 project "COSoDIS – Contract-Oriented Software Development for Internet Services". (http://www.ifi.uio.no/cosodis/)

[1] See technical report [18] for proofs and more explanations.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 494–508, 2010.

Work has been done on defining temporal logics over Mazurkiewicz traces [9] and strong results like decidability and expressive completeness are known [5,20]. For general partial orders, temporal logics usually become undecidable [2]. For the more expressive event structures there are fewer works; a modal logic is investigated in [6].

There is hardly any work on logics for higher dimensional automata [17] and, as far as we know, there is no work on *modal logics for HDAs*. In practice, one is more comfortable with modal logics, like temporal logics or dynamic logics, because these are generally decidable (as opposed to full first-order logic, which is undecidable).

That is why in this paper we introduce and develop a logic in the style of standard modal logic. This logic has *HDAs* as models, hence, the name *higher dimensional modal logic* (*HDML*). This is our basic language to talk about general models of concurrent systems. For this basic logic we prove decidability using a filtration argument. Also, we provide an axiomatic system and prove it is sound and complete for the higher dimensional automata. *HDML* in its basic variant is shown to become standard modal logic when the language and the higher dimensional models are restricted in a certain way.

HDML contrasts with standard temporal/modal logics in the fact that *HDML* can reason about *what holds "during" the execution of some concurrent events.* The close related logic for distributed transition systems of [7] is in the same style of reasoning only about what holds "after" some concurrent events have finished executing. As we show in Section 3, the "after" logics can be encoded in *HDML*, hence also the logic of [7].

Another purpose of this work is to provide a general framework for reasoning about concurrent systems at any level of abstraction and granularity, accounting also for choices and independence of actions. Thus, the purpose of Section 3 is to show that studying *HDML*, and particular variants of it, is fruitful for analyzing concurrent systems and their logics. In this respect we study variants of higher dimensional modal logic inspired by temporal logic and dynamic logic. Already in Section 3.2 we add to the basic language an *Until* operator, in the style of temporal logics. We show how this variant of *HDML*, when interpreted over the class of *HDAs* corresponding to Kripke structures, can be particularized to LTL [11]. A second variant, in Section 3.3, decorates the *HDML* modalities with labels. This multi-modal variant of *HDML* together with the *Until* operator, when interpreted over the class of *HDAs* that encodes Mazurkiewicz traces, becomes LTrL [20] (the linear time temporal logic over Mazurkiewicz traces).

2 Modal Logic over Higher Dimensional Automata

We define higher dimensional automata (*HDAs*) following the definition and terminology of [21,17]. Afterwards, we propose *higher dimensional modal logic* (*HDML*) for reasoning about concurrent systems modeled as *HDAs*. The semantic interpretation of the language is defined in terms of *HDAs* (i.e., the *HDAs*, with a valuation function attached, are the models we propose for *HDML*).

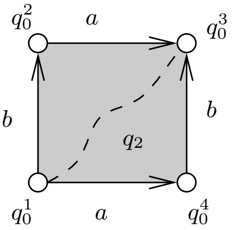


Fig. 1. Example of a *HDA* with two concurrent events

For an intuitive understanding of the *HDA* model consider the standard example [17,21] pictured in Figure 1. It represents a *HDA* that models two concurrent events which are labeled by a and b (one might have the same label a for both events). The *HDA* has four states, q_0^1 to q_0^4, and four transitions between them. This would be the standard picture for interleaving, but in the case of *HDA* there is also a square q_2. Traversing through the interior of the square means that both events are executing. When traversing on the lower transition means that event a is executing but event b has not started yet, whereas, when traversing through the upper transition it means that event a is executing and event b has finished already. In the states there is no event executing, in particular, in state q_0^3 both events have finished, whereas in state q_0^1 no event has started yet.

Similarly, *HDAs* allow one to represent three concurrent events through a cube, or more events through hypercubes. Causality of events is modeled by sticking such hypercubes one after the other. For our example, if we omit the interior of the square (i.e., the grey q_2 is removed) we are left with a description of a system where there is the choice between two sequences of two events, i.e., $a; b + b; a$.

Definition 2.1 (higher dimensional automata). *A* cubical set $H = (Q, \overline{s}, \overline{t})$ *is formed of a family of sets* $Q = \bigcup_{n=0}^{\infty} Q_n$ *with all sets* Q_n *disjoint, and for each* n*, a family of maps* $s_i, t_i : Q_n \to Q_{n-1}$ *with* $1 \leq i \leq n$ *which respect the following* cubical laws*:*

$$\alpha_i \circ \beta_j = \beta_{j-1} \circ \alpha_i, \quad 1 \leq i < j \leq n \text{ and } \alpha, \beta \in \{s, t\}. \tag{1}$$

In H*, the* $\overline{s}$ *and* $\overline{t}$ *denote the collection of all the maps from all the families (i.e., for all* n*). A* higher dimensional structure $(Q, \overline{s}, \overline{t}, l)$ *over an alphabet* Σ *is a cubical set together with a* labeling function $l : Q_1 \to \Sigma$ *which respects* $l(s_i(q)) = l(t_i(q))$ *for all* $q \in Q_2$ *and* $i \in \{1, 2\}$.[2] *A* higher dimensional automaton $(Q, \overline{s}, \overline{t}, l, I, F)$ *is a higher dimensional structure with two designated sets of* initial *and* final cells $I \subseteq Q_0$ *and* $F \subseteq Q_0$*.*

We call the elements of Q_0, Q_1, Q_2, Q_3 respectively *states*, *transitions*, *squares*, and *cubes*, whereas the general elements of Q_n are called n-dimensional cubes (or

[2] Later, in Definition 3.7, the labeling is extended naturally to all cells.

hypercubes). We call generically an element of Q a *cell* (also known as n-cell). For a transition $q \in Q_1$ the $s_1(q)$ and $t_1(q)$ represent respectively its source and its target cells (which are *states* from Q_0 in this case). Similarly for a general cell $q \in Q_n$ there are n source cells and n target cells all of dimension $n-1$. Intuitively, an n-dimensional cell q represents a configuration of a concurrent system in which n events are performed at the same time, i.e., concurrently. A source cell $s_i(q)$ represents the configuration of the system before the starting of the i^{th} event, whereas the target cell $t_i(q)$ represents the configuration of the system immediately after the termination of the i^{th} event. A transition of Q_1 represents a configuration of the system in which a single event is performed.

The cubical laws account for the geometry (concurrency) of the *HDAs*; there are four kinds of cubical laws depending on the instantiation of α and β. For the example of Figure 1 consider the cubical law where α is instantiated to t and β to s, and $i = 1$ and $j = 2$: $t_1(s_2(q_2)) = s_1(t_1(q_2))$. In the left hand side, the second source cell of q_2 is, in this case, the transition $s_2(q_2) = q_1^1 = (q_0^1, q_0^2)$ and the first target cell of q_1^1 is q_0^2 (the only target cell because $s_2(q_2) \in Q_1$); this must be the same cell when taking the right hand side of the cubical law, i.e., the first target cell is $t_1(q_2) = q_1^2 = (q_0^2, q_0^3)$ and the first source of q_1^2 is q_0^2.

We propose the language of higher dimensional modal logic for talking about concurrent systems. *HDML* follows the style of standard modal languages [3].

Definition 2.2 (higher dimensional modal logic). *A formula φ in higher dimensional modal logic is constructed using the grammar below, from a set Φ_B of atomic propositions, with $\phi \in \Phi_B$, which are combined using the propositional symbols $\perp$ and $\rightarrow$ (from which all other standard propositional operations are generated), and using the modalities $\{\}$ and $\langle\rangle$.*

$$\varphi \ := \ \phi \mid \perp \mid \varphi \rightarrow \varphi \mid \{\}\varphi \mid \langle\rangle\varphi$$

We call $\{\}$ the *during modality* and $\langle\rangle$ the *terminate modality.* The intuitive reading of $\{\}\varphi$ is: "pick some event from the ones currently not running (must exist at least one not running) and start it; in the new configuration of the system (during which, one more event is concurrently executing) the formula φ must hold". The intuitive reading of $\langle\rangle\varphi$ is: "pick some event from the ones currently running concurrently (must exist one running) and terminate it; in the new configuration of the system the formula φ must hold". This intuition is formalized in the semantics of *HDML.*

The choice of our notation is biased by the intuitive usage of these modalities where the terminate modality talks about what happens after some event is terminated; in this respect being similar to the standard diamond modality of dynamic logic. Later, in Section 3.3, these modalities are decorated with labels. The during modality talks about what happens during the execution of some event and hence we adopt the notation of Pratt [12].

The models of *HDML* are higher dimensional structures together with a valuation function $\mathcal{V} : Q \rightarrow 2^{\Phi_B}$ which associates a set of atomic propositions to each cell (of any dimension). This means that $\mathcal{V}$ assigns some propositions to each state of dimension 0, to each transition of dimension 1, to each square of

Table 1. Semantics for *HDML*

$\mathcal{H}, q \models \phi$	iff $\phi \in \mathcal{V}(q)$. $\quad \mathcal{H}, q \not\models \bot$
$\mathcal{H}, q \models \varphi_1 \rightarrow \varphi_2$	iff when $\mathcal{H}, q \models \varphi_1$ then $\mathcal{H}, q \models \varphi_2$.
$\mathcal{H}, q \models \{\}\varphi$	iff assuming $q \in Q_n$ for some n, $\exists q' \in Q_{n+1}$ s.t. $s_i(q') = q$ for some $1 \leq i \leq n$, and $\mathcal{H}, q' \models \varphi$.
$\mathcal{H}, q \models \langle\rangle\varphi$	iff assuming $q \in Q_n$ for some n, $\exists q' \in Q_{n-1}$ s.t. $t_i(q) = q'$ for some $1 \leq i \leq n$, and $\mathcal{H}, q' \models \varphi$.

dimension 2, to each cube of dimension 3, etc. Denote a model of *HDML* by $\mathcal{H} = (Q, \overline{s}, \overline{t}, l, \mathcal{V})$. A *HDML* formula is evaluated in a cell of such a model $\mathcal{H}$.

One may see the *HDML* models as divided into *levels*, each level increasing the concurrency complexity of the system; i.e., level Q_n increases the complexity compared to level Q_{n-1} by adding one more event (to have n events executing concurrently instead of $n-1$). The levels are linked together through the s_i and t_i maps. With this view in mind the during and terminate modalities should be understood as jumping from one level to the other; the $\{\}$ modality jumps one level up, whereas the $\langle\rangle$ modality jumps one level down.

Definition 2.3 (satisfiability). *Table 1 defines recursively the satisfaction relation $\models$ of a formula φ w.r.t. a model $\mathcal{H}$ in a particular n-cell q (for some arbitrary n); denote this as $\mathcal{H}, q \models \varphi$. The notions of satisfiability and validity are defined as usual.*

Both modalities have an existential flavor. In particular note that $\mathcal{H}, q_0 \not\models \langle\rangle\varphi$, for $q_0 \in Q_0$ a state, because there is no event executing in a state, and thus no event can be terminated. Similarly, for the during modality, $\mathcal{H}, q_n \not\models \{\}\varphi$ for any n-cell $q_n \in Q_n$ when all sets Q_k, with $n < k$, are empty (i.e., the family of sets Q is bounded by n). This says that there can be at most n events running at the same time, and when reaching this limit one cannot start another event and therefore $\{\}\varphi$ cannot be satisfied.

The universal correspondents of $\{\}$ and $\langle\rangle$ are defined in the standard style of modal logic. We denote these modalities by respectively $\{\![\,]\!\}\varphi$ and $[\,]\varphi$. The intuitive reading of $[\,]\varphi$ is: "pick any of the events currently running concurrently and after terminating it, φ must hold in the new configuration of the system". Note that this modality holds trivially for any state $q_0 \in Q_0$, i.e., $\mathcal{H}, q_0 \models [\,]\varphi$.

2.1 Decidability and Completeness

In the rest of this section we prove that satisfiability for *HDML* is decidable using the filtration technique [3]. Then we give an axiomatic system for *HDML* and prove its soundness and completeness. Completeness is based on constructing canonical models.

The filtration for the states is the same as in the standard modal logic, but for cells of dimension at least 1 we need to take care that the maps t and s in the filtration model remain maps and that they respect the cubical laws so that the filtration is still a *HDML* model. This can be done, but the filtration model is bigger than what is obtained in the case of standard modal logic. On top, the proof of the small model property (Theorem 2.10) is more involved due to the complexities of the definition of filtration given in Definition 2.5.

Definition 2.4 (subformula closure). *The* subformula closure *of a formula φ is the set of formulas $\mathcal{C}(\varphi)$ defined recursively as:*

$$\begin{aligned}
\mathcal{C}(\phi) &\triangleq \{\phi\}, \textit{ for } \phi \in \Phi_B\\
\mathcal{C}(\varphi_1 \to \varphi_2) &\triangleq \{\varphi_1 \to \varphi_2\} \cup \mathcal{C}(\varphi_1) \cup \mathcal{C}(\varphi_2)\\
\mathcal{C}(\{\}\varphi) &\triangleq \{\{\}\varphi\} \cup \mathcal{C}(\varphi)\\
\mathcal{C}(\langle\rangle\varphi) &\triangleq \{\langle\rangle\varphi\} \cup \mathcal{C}(\varphi)
\end{aligned}$$

The *size* of a formula (denoted $|\varphi|$) is calculated by summing the number of Boolean and modal symbols with the number of atomic propositions and $\bot$ symbols that appear in the formula. (All instances of a symbol are counted.) The size of the subformula closure is linear in the size of the formula, $|\mathcal{C}(\varphi)| \leq |\varphi|$.

Definition 2.5 (filtration). *Given a formula φ, we define below a relation $\equiv$ (which is easily proven to be an equivalence relation) over the cells of a higher dimensional structure $\mathcal{H}$, where $q, q' \in Q_i$, for some $i \in \mathbb{N}$:*

$$q \equiv q' \textit{ iff for any } \psi \in \mathcal{C}(\varphi) \textit{ then } (\mathcal{H}, q \models \psi \textit{ iff } \mathcal{H}, q' \models \psi).$$

A filtration $\mathcal{H}^f = (Q^f, s^f, t^f, l^f, \mathcal{V}^f)$ *of* $\mathcal{H}$ *through the closure set* $\mathcal{C}(\varphi)$ *is as below:*

$$\begin{aligned}
Q^f_n &\triangleq \{[q_n] \mid q_n \in Q_n\}, \textit{ where } [q_n] \textit{ is}\\
[q_0] &\triangleq \{q' \mid q_0 \equiv q'\} \textit{ when } q_0 \in Q_0, \textit{ otherwise,}\\
[q_n] &\triangleq \{q' \mid q_n \equiv q' \wedge t_i(q') \in [p_i] \wedge s_i(q') \in [p'_i]\\
&\qquad \textit{for all } 1 \leq i \leq n \textit{ and for some fixed } [p_i], [p'_i] \in Q^f_{n-1}\}.\\
s^f_i([q_n]) &\triangleq [q_{n-1}] \;\; \textit{iff for all } p \in [q_n],\ s_i(p) \in [q_{n-1}].\\
t^f_i([q_n]) &\triangleq [q_{n-1}] \;\; \textit{iff for all } p \in [q_n],\ t_i(p) \in [q_{n-1}].\\
\mathcal{V}^f([q]) &\triangleq \mathcal{V}(q).
\end{aligned}$$

Proposition 2.6 (filtration is a model). *The filtration $\mathcal{H}^f$ of a model $\mathcal{H}$ through a closure set $\mathcal{C}(\varphi)$ is a higher dimensional structure (i.e., is still a HDML model).*

Proof (sketch). Essentially, the proof amounts to showing that the definitions of s^f_i and t^f_i are that of *maps* (as required in a higher dimensional structure) and that they respect the *cubical laws* (see full proof in [18]).

Lemma 2.7 (sizes of filtration sets). *Each set Q^f_n of the filtration $\mathcal{H}^f$ obtained in Definition 2.5 has finite size which depends on the size of the formula φ used in the filtration; more precisely each Q^f_n is bounded from above by $2^{|\varphi| \cdot N}$ where $N = n! \cdot \sum_{k=0}^{n} \frac{2^k}{(n-k)!}$.*

Lemma 2.8 (filtration lemma). *Let $\mathcal{H}^f$ be the filtration of $\mathcal{H}$ through the closure set $\mathcal{C}(\varphi)$, as in Definition 2.5. For any formula $\psi \in \mathcal{C}(\varphi)$ and any cell $q \in \mathcal{H}$, we have $\mathcal{H}, q \models \psi$ iff $\mathcal{H}^f, [q] \models \psi$.*

We define two *degrees of concurrency* of a formula φ: the *upwards concurrency* (denoted $|\varphi|_{uc}$) and *downwards concurrency* (denoted $|\varphi|_{dc}$). The degree of upwards concurrency counts the maximum number of nestings of the during modality $\{\}$ that are not compensated by a $\langle\rangle$ modality. (E.g., the formula $\{\}\{\}\phi \vee \{\}\phi'$ has the degree of upwards concurrency equal to 2, the same as $\{\}\langle\rangle\{\}\{\}\phi$.) The formal definition of $|\;|_{uc}$ is:

$$\begin{aligned}
|\bot|_{uc} &\triangleq |\phi|_{uc} \triangleq 0, \text{ for } \phi \in \Phi_B\\
|\varphi_1 \rightarrow \varphi_2|_{uc} &\triangleq max(|\varphi_1|_{uc}, |\varphi_2|_{uc})\\
|\{\}\varphi|_{uc} &\triangleq 1 + |\varphi|_{uc}\\
|\langle\rangle\varphi|_{uc} &\triangleq max(0, |\varphi|_{uc} - 1)
\end{aligned}$$

The definition of the degree of downwards concurrency $|\;|_{dc}$ is symmetric to the one above in the two modalities; i.e., interchange the modalities in the last two lines. Note that $|\varphi|_{uc} + |\varphi|_{dc} \leq |\varphi|$. The next result offers a safe reduction of a model where we remove all cells which have dimension greater than some constant depending on the formula of interest.

Lemma 2.9 (concurrency boundedness). *If a HDML formula φ is satisfiable, $\mathcal{H}, q \models \varphi$ with $q \in Q_k$, then it exists a model with all the sets Q_m, with $m > |\varphi|_{uc} + k$, empty, which satisfies the formula.*

Notation: The formula $\langle\rangle\phi \wedge \langle\rangle\neg\phi$ expresses that there can be terminated at least two different events (in other words, the cell in which the formula is evaluated to true has dimension at least two). Similarly the formula $\langle\rangle(\phi \wedge \neg\phi') \wedge \langle\rangle(\neg\phi \wedge \neg\phi') \wedge \langle\rangle(\neg\phi \wedge \phi')$ says that there are at least three events that can be terminated. For each $i \in \mathbb{N}^*$ one can write such a formula to say that there are at least i events that can be terminated. Denote such a formula by $\langle\rangle i$. Also define $\langle\rangle^i\varphi$ as i applications of the $\langle\rangle$ modality to φ (i.e., $\langle\rangle \ldots \langle\rangle\varphi$ where $\langle\rangle$ appears i times). Similar, for the during modality denote $\{\}i$ the formula that can start i different events, and by $\{\}^i\varphi$ the i applications of $\{\}$ to φ.

Theorem 2.10 (small model property). *If a HDML formula φ is satisfiable then it is satisfiable on a finite model with no more than $\sum_{n=0}^{|\varphi|} 2^{|\varphi| \cdot N}$ cells where $N = n! \cdot \sum_{k=0}^{n} \frac{2^k}{(n-k)!}$.*

Proof. Note first that it is easy to prove that any formula $\langle\rangle i \rightarrow \langle\rangle^i\top$ is valid, for any $i \in \mathbb{N}^*$. Because of this, the downwards concurrency measure for a formula may be misleading as $|\langle\rangle i|_{dc} = 1$ whereas $|\langle\rangle^i\top|_{dc} = i$. On the other hand the dimension $|\langle\rangle i|$ grows faster than linear with i. It is easy to see that $|\langle\rangle i| > |\langle\rangle^i\top|_{dc}$. If our formula φ has subformulas of the kind $\langle\rangle i$ then the measure $|\varphi|_{dc}$ must be adjusted accordingly. In any case, it is clear that even after adjustment $|\varphi|_{dc} \leq |\varphi| - |\varphi|_{uc}$.

Assume that there exists a model $\mathcal{H}$ and a cell $q \in Q_l$ in this model for which $\mathcal{H}, q \models \varphi$. We can prove, analogous to the proof of Lemma 2.9, that for a formula φ one needs to look at cells of dimension at least $|\varphi|_{dc}$. A more coarse approximation is to say that one needs all the sets Q_n with $n \leq |\varphi| - |\varphi|_{uc}$. Thus, we can safely assume $l \leq |\varphi| - |\varphi|_{uc}$.

Table 2. Axiomatic system for *HDML*

Axiom schemes:

(A1) All instances of propositional tautologies.

(A2) $\{\}\bot \leftrightarrow \bot$	(A2') $\langle\rangle\bot \leftrightarrow \bot$
(A3) $\{\}(\varphi \vee \varphi') \leftrightarrow \{\}\varphi \vee \{\}\varphi'$	(A3') $\langle\rangle(\varphi \vee \varphi') \leftrightarrow \langle\rangle\varphi \vee \langle\rangle\varphi'$
(A4) $\{\!\![\,]\!\!\}\varphi \leftrightarrow \neg\{\}\neg\varphi$	(A4') $[\,]\varphi \leftrightarrow \neg\langle\rangle\neg\varphi$
(A5) $\langle\rangle i \rightarrow \langle\rangle^i\top \quad \forall i \in \mathbb{N}^*$	(A5') $\{\}^i\top \rightarrow \{\}i \quad \forall i \in \mathbb{N}^*$
(A6) $\langle\rangle[\,]\varphi \rightarrow [\,]\langle\rangle\varphi$	(A6') $\{\}\{\!\![\,]\!\!\}\varphi \rightarrow \{\!\![\,]\!\!\}\{\}\varphi$
(A7) $\{\}[\,]\varphi \rightarrow [\,]\{\}\varphi$	(A7') $\langle\rangle\{\!\![\,]\!\!\}\varphi \rightarrow \{\!\![\,]\!\!\}\langle\rangle\varphi$
(A8) $\{\}\langle\rangle^i\top \rightarrow \{\!\![\,]\!\!\}\langle\rangle^i\top \quad \forall i \in \mathbb{N}$	(A8') $\langle\rangle\langle\rangle^i\top \rightarrow [\,]\langle\rangle^i\top \quad \forall i \in \mathbb{N}$
(A9) $\langle\rangle^i\top \rightarrow \{\!\![\,]\!\!\}\langle\rangle\langle\rangle^i\top \quad \forall i \in \mathbb{N}$	(A9') $\{\}\langle\rangle\langle\rangle^i\top \rightarrow \langle\rangle^i\top \quad \forall i \in \mathbb{N}$

Inference rules:

(R1) $\dfrac{\varphi \qquad \varphi \rightarrow \varphi'}{\varphi'}$ (MP) (R2) $\dfrac{\varphi \rightarrow \varphi'}{\{\}\varphi \rightarrow \{\}\varphi'}$ (D) (R2') $\dfrac{\varphi \rightarrow \varphi'}{\langle\rangle\varphi \rightarrow \langle\rangle\varphi'}$

(R3) Uniform variable substitution.

From Lemma 2.9 we know that we need to consider only the sets Q_m for $m \leq l + |\varphi|_{uc} \leq |\varphi|$, and all other sets Q are empty. From Lemma 2.8 we know that we can build a filtration model $\mathcal{H}^f$ s.t. the formula φ is still satisfiable and, by Lemma 2.7, we know that all the sets Q^f_m have a finite number of cells. Thus we can sum up all the cells in all the Q^f_m, with $m \leq |\varphi|$.

Corollary 2.11 (decidability). *Deciding the satisfiability of a HDML formula φ is done in space at most $\sum_{n=0}^{|\varphi|} 2^{|\varphi|\cdot N}$ where N is defined in Theorem 2.10.*

In the following we go on giving an axiomatic system for *HDML* and prove it sound and complete. In Table 2 we give a set of axioms and rules of inference for *HDML*. If a formula is *derivable* in this axiomatic system we write $\vdash \varphi$. We say that a formula φ is derivable from a set of formulas S iff $\vdash \psi_1 \wedge \cdots \wedge \psi_n \rightarrow \varphi$ for some $\psi_1, \ldots, \psi_n \in S$ (we write equivalently $S \vdash \varphi$). A set of formulas S is said to be *consistent* if $S \not\vdash \bot$, otherwise it is said to be *inconsistent*. A consistent set S is called *maximal* iff all sets S', with $S \subset S'$, are inconsistent.

Theorem 2.12 (soundness). *The axiomatic system of Table 2 is sound. Formally $\forall\varphi : \vdash \varphi \Rightarrow \models \varphi$.*

Proof (sketch). The proof tests that all axioms are valid and that all inference rules preserve validity. We check only the non-standard axioms (A5) to (A9').

We fix some terminology and notation. Denote by $\neg\mathcal{C}(\varphi) = \mathcal{C}(\varphi) \cup \{\neg\varphi' \mid \varphi' \in \mathcal{C}(\varphi)\}$ the set of subformulas, as in Definition 2.4, together with their negated forms. A set of formulas A is called an *atom* for φ if A is a maximal consistent subset of $\neg\mathcal{C}(\varphi)$. Denote by $At(\varphi)$ the set of all atoms for φ.

Definition 2.13 (canonical saturated *HDA*). *A HDA is called* canonical for the formula φ *if a* canonical labeling $\lambda : Q \rightarrow At(\varphi)$ *can be attached to the HDA. A labeling is* canonical *if the following conditions hold:*

1. *for any* $q_n \in Q_n, q_{n-1} \in Q_{n-1}$*, for some* $n > 0$*, and* $\forall 0 \leq i \leq n$*, if* $s_i(q_n) = q_{n-1}$ *then* $\forall \psi \in \neg\mathcal{C}(\varphi)$ *if* $\psi \in \lambda(q_n)$ *then* $\{\}\psi \in \lambda(q_{n-1})$*,*
2. *for any* $q_n \in Q_n, q_{n-1} \in Q_{n-1}$*, for some* $n > 0$*, and* $\forall 0 \leq i \leq n$*, if* $t_i(q_n) = q_{n-1}$ *then* $\forall \psi \in \neg\mathcal{C}(\varphi)$ *if* $\psi \in \lambda(q_{n-1})$ *then* $\langle\rangle\psi \in \lambda(q_n)$*.*

A canonical HDA is called saturated *if:*

1. *whenever* $\{\}\psi \in \lambda(q_{n-1})$ *then* $\exists q_n \in Q_n$ *and* $\exists 0 \leq i \leq n$ *s.t.* $s_i(q_n) = q_{n-1}$ *and* $\psi \in \lambda(q_n)$*,*
2. *whenever* $\langle\rangle\psi \in \lambda(q_n)$ *then* $\exists q_{n-1} \in Q_{n-1}$ *and* $\exists 0 \leq i \leq n$ *s.t.* $t_i(q_n) = q_{n-1}$ *and* $\psi \in \lambda(q_{n-1})$*.*

Lemma 2.14 (truth lemma). *In a canonical saturated HDA* $\mathcal{H}$ *with the valuation defined as* $\mathcal{V}(q_n) = \{\phi \in \Phi_B \mid \phi \in \lambda(q_n)\}$*, it holds that* $\mathcal{H}, q_n \models \psi$ *iff* $\psi \in \lambda(q_n)$*.*

To prove completeness of the axiomatic system all that remains is to show that for any consistent formula φ we can build such a canonical saturated *HDA*. While building the canonical saturated *HDA* we constantly struggle to saturate the *HDA* (that we work with) while respecting the canonicity. Such not saturated *HDAs* are called *defective*, as they may have defects, which we formally define below. But important is that any of these defects can be repaired. This is what the repair lemma does, using the two *enriching* and *lifting* constructions. The completeness theorem then shows that while starting with a minimal canonical *HDA* we can incrementally build a defect free canonical *HDA*.

Definition 2.15 (defects). *There are two types of defects for* $\mathcal{H}$ *(each corresponding to a violation of a saturation condition):*

- *a* D1 defect *of* $\mathcal{H}$ *is a cell* $q_n \in Q_n$ *with* $\{\}\psi \in \lambda(q_n)$ *for which there is no* $q_{n+1} \in Q_{n+1}$ *and no* $1 \leq i \leq n+1$*, with* $s_i(q_{n+1}) = q_n$ *and* $\psi \in \lambda(q_{n+1})$*;*
- *a* D2 defect *of* $\mathcal{H}$ *is a cell* $q_n \in Q_n$ *with* $\langle\rangle\psi \in \lambda(q_n)$ *for which there is no* $q_{n-1} \in Q_{n-1}$ *and no* $1 \leq i \leq n-1$*, with* $t_i(q_n) = q_{n-1}$ *and* $\psi \in \lambda(q_{n-1})$*.*

For two *HDAs*, $\mathcal{H}_1$ and $\mathcal{H}_2$ we say that $\mathcal{H}_2$ *extends* $\mathcal{H}_1$ (written $\mathcal{H}_2 \rhd \mathcal{H}_1$) iff $\mathcal{H}_2$ has all the cells (with the same labels) and maps of $\mathcal{H}_1$ and possibly some new cells and maps (i.e., some extra structure).

Lemma 2.16 (enriching construction). *For a canonical model* $\mathcal{H}$*, there exists a construction (see [18]),called* enriching of the $\mathcal{H}$ w.r.t. q and a formula $\{\}\varphi \in \lambda(q)$*, which builds a model* $\mathcal{H}' \rhd \mathcal{H}$ *that is canonical and extends* $\mathcal{H}$*.*

The *enriching construction* adds one new cell that has q as one of its sources and is labeled with an atom containing φ. Moreover, all the other maps of this new cell need to be added, together with all the necessary new cells, taking care to respect the cubical laws.

The *lifting construction* lifts all the cells of each level one level up by adding one new s and t map to each. The cubical laws make sure that these new maps reach only new cells; none of the old cells (that are lifted) are involved in these new instances of the cubical laws. We need to be careful how we label all these new cells s.t. the canonicity is respected for the extended $\mathcal{H}'$.

Lemma 2.17 (lifting construction). *For a canonical model $\mathcal{H}$, there exists a construction (see [18]), called* lifting of the $\mathcal{H}$ w.r.t. q and a formula $\langle\rangle\varphi \in \lambda(q)$, *which builds a model $\mathcal{H}' \rhd \mathcal{H}$ that is canonical and extends $\mathcal{H}$.*

Lemma 2.18 (repair lemma). *For any canonical HDA $\mathcal{H}$ that has a defect we can build a corresponding $\mathcal{H}'$ which is canonical and does not have this defect.*

Proof (sketch). We repair a D1 (respectively D2) defect using the *enriching* (respectively *lifting*) construction.

Theorem 2.19 (completeness). *The axiomatic system of Table 2 is complete. Formally $\forall\varphi : \models \varphi \;\Rightarrow\; \vdash \varphi$.*

Proof. Using the truth lemma 2.14, the proof amounts to showing that for any consistent formula φ we can build a canonical saturated $\mathcal{H}_\varphi$ that has a cell labeled with an atom that contains φ. We construct $\mathcal{H}_\varphi$ in steps starting with $\mathcal{H}^0_\varphi$ which contains only one cell q^0_0 of dimension 0 labeled with an atom containing φ, i.e., $\lambda(q^0_0) = A_\varphi$. Trivially, $\mathcal{H}^0_\varphi$ is canonical. The cells used to construct our model are picked (in the right order) from the following sets $S_i = \{q^j_i \mid j \in \omega\}$ where $i \in \omega$ corresponds to the dimension i. Any of these cells may have defects and thus, we list all the defects, i.e., all the cells, and try to repair them in increasing order (i.e., we treat first defects on level 0 and continue upwards).

At some step $n \geq 0$ in the construction we consider $\mathcal{H}^n_\varphi = (Q^n, \overline{s^n}, \overline{t^n}, l^n)$ canonical. If $\mathcal{H}^n_\varphi$ is not saturated then pick the smallest defect cell of $\mathcal{H}^n_\varphi$. For a D1 defect, i.e., a cell $q_k \in Q_k$ and formula $\{\}\psi \in \lambda(q_k)$, apply enrich(k,$q_k$,$\psi$) and obtain a model $\mathcal{H}^{n+1}_\varphi$ which is canonical, cf. Lemma 2.16, and does not have the D1 defect, cf. Lemma 2.18. For a D2 defect apply the lifting construction to remove the defect. Moreover, any repaired defect will never appear in any extension model, independent of how many times we apply the enriching or lifting constructions. Both enriching and lifting pick their new cells from S in increasing order. We obtain $\mathcal{H}_\varphi$ as a limit construction from all the $\mathcal{H}^n_\varphi$; i.e., $\mathcal{H}_\varphi = (Q, \overline{s}, \overline{t}, l)$ as $Q = \bigcup_{n\in\omega} Q^n,\;\; \overline{s} = \bigcup_{n\in\omega} \overline{s^n},\;\; \overline{t} = \bigcup_{n\in\omega} \overline{t^n},\;\; l = \bigcup_{n\in\omega} l^n$.

3 Encodings into Higher Dimensional Modal Logic

This section serves to exemplify two main ways of using *HDML*. One usage is as a highly expressive logic in which many other logics for different concurrency models can be encoded; in this respect we study the relation of *HDML* with standard modal logic, LTL, and linear time temporal logic over Mazurkiewicz traces LTrL. The other usage is as a general theoretical framework for studying different logics for different concurrency models (that can be expressed as some class of *HDA*) and their interrelation. This is done by finding the apropriate restrictions of *HDA* and *HDML* and investigating their relations in *HDML*.

3.1 Encoding Standard Modal Logic into HDML

Lemma 3.1 (Kripke structures). *The class of Kripke structures is captured by the class of higher dimensional structures where all sets Q_n, for $n > 1$, are empty.*

Proof. Essentially this result is found in [21]. A *HDA* $K = (Q_0, Q_1, s_1, t_1, l)$ is a special case of *HDAs* where all $Q_n = \emptyset$ for $n > 1$. This is the class of *HDAs* that encode Kripke frames. Because Q_2 (and all other cells of higher dimension) is empty there are no cubical laws applicable. Therefore, there is no geometric structure on K. Moreover, the restriction on the labeling function l is not applicable (as Q_2 is empty). Add to such a *HDA* a valuation function $\mathcal{V}$ to obtain a Kripke model $(Q_0, Q_1, s_1, t_1, l, \mathcal{V})$.

Proposition 3.2 (axiomatization of Kripke *HDAs*). *The class of higher dimensional structures corresponding to Kripke structures (from Lemma 3.1) is axiomatized by:*

$$\models \neg\langle\rangle\top \rightarrow \lBrace\rBrace\lBrace\rBrace\bot$$

Proof (sketch). For any *HDA* $\mathcal{H}$ and any $q \in Q$ a cell of any dimension, we prove the double implication: $\mathcal{H}, q \models \neg\langle\rangle\top \rightarrow \lBrace\rBrace\lBrace\rBrace\bot$ iff $\mathcal{H}$ is as in Lemma 3.1.

Theorem 3.3 (standard modal logic). *Let the syntactic construct $\Diamond\varphi \triangleq \{\}\langle\rangle\varphi$. The language of* standard modal logic *uses only $\Diamond$ and is interpreted only over higher dimensional structures as defined in Lemma 3.1 and only in cells of Q_0.*

Proof (sketch). First we check that we capture exactly the semantics of standard modal logic. Second we check that we recover the axiomatic system of standard modal logic for $\Diamond$ from the axiomatic system of *HDML*.

3.2 Adding an Until Operator and Encoding LTL

The basic temporal logic is the logic with only the *eventually* operator (and the dual *always*). This language is expressible in the standard modal logic [3] and thus is expressible in *HDML* too. It is known that the *Until* operator $\mathcal{U}$ adds expressivity to LTL (*eventually* and *always* operators can be encoded with $\mathcal{U}$ but not the other way around).

The *Until* operator cannot be encoded in *HDML* because of the local behavior of the during and terminate modalities; similar arguments as in modal logic about expressing *Until* apply to *HDML* too. The *Until* modality talks about the whole model (about all the configurations of the system) in an existential manner. More precisely, the *Until* says that there must exist some configuration in the model, reachable from the configuration where *Until* is evaluated, satisfying some property φ and in all the configurations on all the paths reaching the φ configuration some other property ψ must hold. Hence we need a notion of *path* in a *HDA*.

Definition 3.4 (paths in ***HDAs*****).** *A* simple step *in a HDA is either* $q_{n-1} \xrightarrow{s_i} q_n$ *with* $s_i(q_n) = q_{n-1}$ *or* $q_n \xrightarrow{t_i} q_{n-1}$ *with* $t_i(q_n) = q_{n-1}$*, where* $q_n \in Q_n$ *and* $q_{n-1} \in Q_{n-1}$ *and* $1 \leq i \leq n$*. A* path $\pi \triangleq q^0 \xrightarrow{\alpha^0} q^1 \xrightarrow{\alpha^1} q^2 \xrightarrow{\alpha^2} \dots$ *is a sequence of single steps* $q^j \xrightarrow{\alpha^j} q^{j+1}$*, with* $\alpha^j \in \{s_i, t_i\}$*. We say that* $q \in \pi$ *iff* $q = q^j$ *appears in one of the steps in* π*. The first cell in a path is denoted* $st(\pi)$ *and the ending cell in a finite path is* $en(\pi)$*.*

In the same spirit as done for temporal logic we boost the expressivity of *HDML* by defining an $\mathcal{U}$ operator over higher dimensional structures. Using this operator we can encode the standard *Until* operator of LTL.

Definition 3.5 (Until operator). *Define an Until operator* $\varphi \mathcal{U} \varphi'$ *which is interpreted over a HDA in a cell as below:*

$$\mathcal{H}, q \models \varphi \mathcal{U} \varphi' \text{ iff } \exists \pi \in \mathcal{H} \text{ s.t. } st(\pi) = q \wedge en(\pi) = q', \\ \mathcal{H}, q' \models \varphi', \text{ and } \forall q'' \in \pi, q'' \neq q' \text{ then } \mathcal{H}, q'' \models \varphi.$$

Proposition 3.6 (modeling LTL). *The LTL Until modality is encoded syntactically by* $\varphi \overline{\mathcal{U}} \varphi' \triangleq (\varphi \vee \langle\rangle\top) \mathcal{U} (\varphi' \wedge \neg\langle\rangle\top)$ *when* $\overline{\mathcal{U}}$ *is interpreted only in states of Kripke HDAs as in Lemma 3.1.*

3.3 Partial Order Models and Their Logics in HDML

This section is mainly concerned with Mazurkiewicz traces [8] as a model of concurrency based on partial orders, because of the wealth of logics that have been developed for it [9,20]. Higher dimensional automata are more expressive than most of the partial orders models (like Mazurkiewicz traces, pomsets [13], or event structures [10]) as studied in [16,21]. The works of [16,17,21] show (similar in nature) how event structures can be encoded in higher dimensional automata. We consider the presentation of Mazurkiewicz traces as a particular class of event structures, precisely defined in [19]. Because of space constraints we cannot give standard definitions and results on partial orders, event structures, or Mazurkiewicz traces; one can check these in [18].

Results from [17] show that a finitary event structure is uniquely determined by its family of configurations, denoted $(E, \mathcal{C}_E)$. One can view a configuration of an event structure as a restricted valuation of events $E \to \{0, 1\}$, and thus we can view an event structure as a valuation $f_E : 2^E \to \{0, 1\}$, which selects only those configurations that make the event structure.

The terminology that we adopt here steams from the Chu spaces representation of *HDAs* [16,17]. We fix a set E, which for our purposes denotes events. Consider the class of *HDAs* which have a single hypercube of dimension $|E|$, hence each event represents one dimension in the *HDA*. This hypercube is denoted 3^E, in relation to 2^E, because in the *HDA* case each event may be in three phases: *not started*, *executing*, and *terminated* (as opposed to only terminated or not started). The valuation from before becomes $E \to \{0, \frac{1}{2}, 1\}$, where $\frac{1}{2}$ means executing. The set of three values is linearly ordered $0 < \frac{1}{2} < 1$ to obtain an *acyclic HDA* [17], and all cells of 3^E (i.e., the configurations) are ordered by the natural lifting of this order pointwise. The dimension of a cell is equal to the number of $\frac{1}{2}$ in its corresponding valuation.

Notation: In the context of a single hypercube 3^E we denote the cells by lists of $|E|$ elements $e_1 e_2 \ldots e_{|E|}$ where each e_i takes values in $\{0, \frac{1}{2}, 1\}$ and represents the status of the i^{th} event of the *HDA*.

With the above conventions, the cells of dimension 0 (i.e., the states of the *HDA*) are denoted by the corresponding valuation restricted to only the two values $\{0,1\}$; and correspond to the configurations of an event structure. The set of states of such a *HDA* is partially ordered by the order $<$ we defined before. In this way, from the hypercube 3^E we can obtain any family of configurations $\mathcal{C}_E$ by removing all 0-dimensional cells that represent a configuration $C \notin \mathcal{C}_E$.[3] By results in [17] we can reconstruct the event structure.

In Definition 2.3 the interpretation of the during and terminate modalities of *HDML* did not take into consideration the labeling of the *HDA*. The labeling was used only for defining the geometry of concurrency of the *HDA*. Now we make use of this labeling function in the semantics of the labeled modalities of Definition 3.8. But first we extend the labeling to cells of any dimension.

Definition 3.7 (general labeling). *Because of the condition $l(s_i(q)) = l(t_i(q))$ for all $q \in Q_2$, all the edges $e_1 \ldots e_{i-1} \frac{1}{2} e_{i+1} \ldots e_{|E|}$, with $e_j \in \{0,1\}$ for $j \neq i$, have the same label. Denote this as the label l_i. The label of a general cell $q \in Q_n$ is the multiset of n labels $l_{j_1} \ldots l_{j_n}$ where the j's are exactly those indexes in the representation of q for which e_j has value $\frac{1}{2}$.*

As is the case with multi-modal logics or propositional dynamic logics, we extend *HDML* to have a multitude of modalities indexed by some alphabet Σ (the alphabet of the *HDA* in our case). This will be the same alphabet as that of the Mazurkiewicz trace represented by the *HDA*.

Definition 3.8 (labeled modalities). *Consider two labeled modalities* during $\{a\}\varphi$ *and* terminate $\langle a\rangle\varphi$ *where $a \in \Sigma$ is a label from a fixed alphabet. The interpretation of the labeled modalities is given below:*

$$\mathcal{H}, q \models \{a\}\varphi \text{ iff assuming } q \in Q_n \text{ for some } n, \exists q' \in Q_{n+1} \text{ s.t. } s_i(q') = q \text{ for some } 1 \leq i \leq n,\ l(q') = l(q)a \text{ and } \mathcal{H}, q' \models \varphi.$$

$$\mathcal{H}, q \models \langle a\rangle\varphi \text{ iff assuming } q \in Q_n \text{ for some } n, \exists q' \in Q_{n-1} \text{ s.t. } t_i(q) = q' \text{ for some } 1 \leq i \leq n,\ l(q) = l(q')a \text{ and } \mathcal{H}, q' \models \varphi.$$

Having the labeled modalities one can get the unlabeled variants as a disjunction over all labels $\{\}\varphi \triangleq \bigvee_{a\in\Sigma}\{a\}\varphi$.

In the remainder of this section we show how the LTrL logic of [20] is captured in the higher dimensional framework. This logic, as well as those presented in [9,4], are interpreted in some particular configuration of a Mazurkiewicz trace (or of a partial order). We take the view of Mazurkiewicz traces as restricted

[3] We remove also all those cells of higher dimension that are connected with the 0-dimensional cells that we have removed.

event structures from, e.g. [9] but we use their representation using their corresponding family of configurations cf. [19,17] (see [18] for details). Therefore, we now interpret *HDML* over restricted *HDAs* as we discussed above.

Proposition 3.9 (encoding LTrL). *The language of LTrL consists of the propositional part of HDML together with the syntactic definitions of the Until operator* $\overline{\mathcal{U}}$ *from Definition 3.6 and* $\overline{\langle a\rangle}\varphi \triangleq \{a\}\langle a\rangle\varphi$ *for* $a \in \Sigma$*. When interpreted only in the states of a HDA representing a Mazurkiewicz trace this language has the same behavior as the one presented in [20]*

Proof. The states of the *HDA* are the configurations of the Mazurkiewicz trace. Thus, our definition of the LTrL language is interpreted in one trace at one particular configuration; as is done in [20]. The original semantics of LTrL uses transitions from one configuration to another labeled by an element from the alphabet Σ of the trace. It is easy to see that our syntactic definition of $\overline{\langle a\rangle}\varphi$ has the same interpretation as the one in [20]. The proof is similar to the proof of Theorem 3.3. The *Until* operator of [20] has the same definition as the one in standard LTL and thus we use the one defined in Proposition 3.6; the proof is easily adapted to the Mazurkiewicz traces setting.

4 Conclusion

We have introduced a modal logic called *HDML* which is interpreted over higher dimensional automata. According to our knowledge, this has not been done before. The language of *HDML* is simple, capturing both the notions of "during" and "after". The associated semantics is intuitive, accounting for the special geometry of the *HDAs*. An adaptation of the filtration method was needed to prove decidability. We have associated to *HDML* an axiomatic system which incorporates the standard modal axioms and has extra only few natural axioms related to the cubical laws and to the dimensions of *HDAs*. This system was proven to be complete for *HDAs*.

We isolated axiomatically the class of *HDAs* that encode Kripke structures and shown how standard modal logic is encoded into *HDML* when interpreted only over these restricted *HDAs*. We then extended the expressiveness of *HDML* by defining an *Until* operator over *HDAs*. Using this *Until* operator, the LTL was encoded into *HDML* when interpreted over the Kripke *HDAs*.

As future work we are investigating a tableaux system for *HDML*. We are also trying to understand better the relation of *HDML* with other logics for weaker models of concurrency like with the modal logic of [6] for event structures or other logics for Mazurkiewicz traces. Particularly interesting is how our results relate to the undecidability results of [2] or to the logic of [1].

Regarding the expressiveness of *HDML*, our current work focuses on finding the kind of bisimulation that is characterized by *HDML* (and its extensions from Section 3). Standard bisimulations for concurrent systems like ST-bisimulation or split-bisimulation are not characterized by *HDML* because of the *during modality* and the "during" behaviour of *HDML*.

Acknowledgements. I thank Martin Steffen and Olaf Owe for comments.

References

1. Alur, R., McMillan, K.L., Peled, D.: Deciding Global Partial-Order Properties. Formal Methods in System Design 26(1), 7–25 (2005)
2. Alur, R., Peled, D.: Undecidability of partial order logics. Inf. Process. Lett. 69(3), 137–143 (1999)
3. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theor. Comput. Sci., vol. 53. Cambridge Univ. Press, Cambridge (2001)
4. Diekert, V., Gastin, P.: LTL Is Expressively Complete for Mazurkiewicz Traces. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 211–222. Springer, Heidelberg (2000)
5. Diekert, V., Gastin, P.: From local to global temporal logics over Mazurkiewicz traces. Theor. Comput. Sci. 356(1-2), 126–135 (2006)
6. Lodaya, K., Mukund, M., Ramanujam, R., Thiagarajan, P.S.: Models and Logics for True Concurrency. Technical Report IMSc-90-12, Inst. Mathematical Science, Madras, India (1990)
7. Lodaya, K., Parikh, R., Ramanujam, R., Thiagarajan, P.S.: A Logical Study of Distributed Transition Systems. Inf. Comput. 119(1), 91–118 (1995)
8. Mazurkiewicz, A.W.: Basic notions of trace theory. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) REX Workshop. LNCS, vol. 354, pp. 285–363. Springer, Heidelberg (1989)
9. Mukund, M., Thiagarajan, P.S.: Linear Time Temporal Logics over Mazurkiewicz Traces. In: Penczek, W., Szałas, A. (eds.) MFCS 1996. LNCS, vol. 1113, pp. 62–92. Springer, Heidelberg (1996)
10. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains. In: Kahn, G. (ed.) Semantics of Concurrent Computation. LNCS, vol. 70, pp. 266–284. Springer, Heidelberg (1979)
11. Pnueli, A.: The temporal logic of programs. In: FOCS 1977, pp. 46–57. IEEE Computer Society Press, Los Alamitos (1977)
12. Pratt, V.R.: A Practical Decision Method for Propositional Dynamic Logic: Preliminary Report. In: STOC 1978, pp. 326–337. ACM Press, New York (1978)
13. Pratt, V.R.: Modeling Concurrency with Partial Orders. J. Parallel Programming 15(1), 33–71 (1986)
14. Pratt, V.R.: Modeling concurrency with geometry. In: POPL 1991, pp. 311–322 (1991)
15. Pratt, V.R.: Chu spaces and their interpretation as concurrent objects. In: van Leeuwen, J. (ed.) Computer Science Today. LNCS, vol. 1000, pp. 392–405. Springer, Heidelberg (1995)
16. Pratt, V.R.: Higher dimensional automata revisited. Math. Struct. Comput. Sci. 10(4), 525–548 (2000)
17. Pratt, V.R.: Transition and Cancellation in Concurrency and Branching Time. Math. Struct. Comput. Sci. 13(4), 485–529 (2003)
18. Prisacariu, C.: Modal Logic over Higher Dimensional Automata – technicalities. Technical Report 393, Univ. Oslo (January 2010)
19. Rozoy, B., Thiagarajan, P.S.: Event structures and trace monoids. Theor. Comput. Sci. 91(2), 285–313 (1991)
20. Thiagarajan, P.S., Walukiewicz, I.: An Expressively Complete Linear Time Temporal Logic for Mazurkiewicz Traces. Information and Computation 179(2), 230–249 (2002)
21. van Glabbeek, R.J.: On the Expressiveness of Higher Dimensional Automata. Theor. Comput. Sci. 356(3), 265–290 (2006)

A Communication Based Model for Games of Imperfect Information

R. Ramanujam[1] and Sunil Simon[2]

[1] The Institute of Mathematical Sciences
C.I.T. Campus, Chennai 600 113, India
jam@imsc.res.in
[2] Institute for Logic, Language and Computation
University of Amsterdam.
s.e.simon@uva.nl

Abstract. The standard way of modelling imperfect information in games is in terms of information partitions for players. In this view, each player is associated with an equivalence relation over the set of game positions. For multiplayer games of imperfect information defined in this manner it turns out that most of the algorithmic questions like determining the winning strategy and synthesis of an equilibrium profile are undecidable. In this light, we propose a model where the players' information partitions are generated explicitly by means of communication. We define a notion of locally consistent equilibrium and suggest that this captures the intuition of stable behaviour of players better. We show that when communication is by means of public announcements, the question of whether locally consistent equilibrium profile exists is decidable.

1 Motivation

Game models have proved to be an attractive way of studying models of reactive systems. Two player turn-based zero-sum games of perfect information provide a natural framework for verification of system properties as well as synthesis of controllers to achieve desired system behaviour. Turn-based games make the strong assumption that players can observe the state of the game and the previous moves before they get their turn to make a move. When these systems are component based, such an assumption is often untenable: typically they exhibit concurrent behaviour and in the natural game model of such a system the moves of one player may well be invisible to others. It can be easily shown that randomized strategies are more powerful than deterministic ones in such situations, and in the setting of concurrent games with win/lose objectives, algorithmic questions on imperfect information have been extensively studied [5,8,4].

In the case of autonomous component based systems, a natural extension of the model involves *non-zero-sum* games where players have preferences over outcomes. Indeed, the classic works of distributed computing such as those on the Byzantine Generals problem [11] envisaged a commonality of concerns between game theory and distributed systems: the presence of uncertainty and

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 509–523, 2010.

agents having possibly different goals who nonetheless need to act together and perhaps coordinate. While these systems may act against global "adversaries", local preferences still matter [9].

This leads us to the setting of games of partial information which are notoriously difficult [13]. Win/lose games are not determined, and in games with preference orderings, many questions are undecidable in the case of reactive systems that involve unbounded play.

A close analysis of the negative results offers us a way out. Two important features of games of partial information are: use of *private communications* and given information partitions. (By this, we mean that in games of imperfect information, players' uncertainty is defined simply by information sets, not in any way determined by other structure in the game.) In this context, concurrency theory, with its rich repertoire of techniques by which uncertainty of agents is structurally determined, is relevant. In models of distributed systems, two global states may look the same to an agent who has the same local state in both, and hence that agent would not be able to observe transitions between such states. Moreover, distributed system theory considers various models of communication.

Such an observation suggests the consideration of games of partial information in which information sets are structurally generated, or means of communication between players rule out private channels. In this paper, we consider a model with both features: where players operate on their own game arenas asynchronously, and hence have only a partial view of global game positions, thus generating information sets. Moreover, all information exchange is via public announcements. We consider only reachability objectives but players have preferences over outcomes. What to communicate to others is a strategic choice exercised by players.

In such a setting, the solution concepts are worth a re-examination as well. Strategies are required to be consistent with information sets and hence are view-based; thus strategies can only depend on players' *observations* of game evolution. It is then natural to consider notions of equilibria where a player's best reponse is not to other players' strategies, but to observations of other players' strategies. We say that a profile is in a locally consistent equilibrium if each player's strategy is symmetrically the best reponse to the observations entailed by other players' strategies.

The main result of the paper states that existence of locally consistent equilibria is decidable in this model of distributed games, where information sets are structurally determined and communication is by way of public announcements. We consider the proposed model and notion of locally consistent equilibrium to be the main contribution of the paper. The results presented here are preliminary and part of ongoing work on characterizing equilibria, on variations in the communication model, and on algorithmic questions on subclasses.

This model should be seen as inspired by, but in contrast with, distributed win/lose games played by asynchronous automata over specific architectures studied by [12,6,1]. A detailed classification of recent work can be found in [7]. Our point of departure is in considering multiplayer non-zero sum games which are structurally generated in terms of the behaviour of players. These games are not necessarily motivated by the distributed synthesis setting and therefore we do not assume the presence of an "environment" player. Other interesting connections relate to logical analyses such as that of *independence-friendly* logics [10], but these are not explored here.

2 Preliminaries

Let $N = \{1, \dots, n\}$ denote the set of players, we use i to range over this set. Let Σ be a finite set of action symbols representing moves of players, we let a, b range over Σ. For a set X and a finite sequence $\varrho = x_1 x_2 \dots x_m \in X^+$, let $last(\varrho) = x_m$ denote the last element in this sequence.

2.1 Game Trees

Let $\mathbb{T} = (S, \Rightarrow, s_0)$ be a tree rooted at s_0 on the set of vertices S and $\Rightarrow : (S \times \Sigma) \to S$ be a *partial* function specifying the edges of the tree. The tree $\mathbb{T}$ is said to be finite if S is a finite set. For a node $s \in S$, let $\overrightarrow{s} = \{s' \in S \mid s \overset{a}{\Rightarrow} s'$ for some $a \in \Sigma\}$ and $\mathsf{d}(s)$ denote the depth of s in $\mathbb{T}$. Let $moves(s) = \{a \in \Sigma \mid \exists s' \in S$ with $s \overset{a}{\Rightarrow} s'\}$. A node s is called a leaf node if $\overrightarrow{s} = \emptyset$.

An extensive form game tree is a pair $T = (\mathbb{T}, \widehat{\lambda})$ where $\mathbb{T} = (S, \Rightarrow, s_0)$ is a tree. The set S denotes the set of game positions with s_0 being the initial game position. The edge function $\Rightarrow$ specifies the moves enabled at a game position and the turn function $\widehat{\lambda} : S \to N$ associates each game position with a player. $T = (\mathbb{T}, \widehat{\lambda})$ is said to be finite if $\mathbb{T}$ is finite. For $i \in N$, let $S^i = \{s \mid \widehat{\lambda}(s) = i\}$ and let *frontier*$(\mathbb{T})$ denote the set of all leaf nodes of T.

A play in T is simply a path $\rho : s_0 a_1 s_1 \cdots$ in $\mathbb{T}$ such that for all $j > 0$, $s_{j-1} \overset{a_j}{\Rightarrow} s_j$. A strategy for player i is a function $\mu^i : S^i \to \Sigma$ which specifies a move at every game position of the player. Let $\Omega^i(T)$ denote the set of all strategies of player i in T. We use $\overline{\mu} = (\mu^1, \dots, \mu^n)$ to denote a profile of strategies. For a player $i \in N$ we also use the notation μ^{-i} to denote the profile $(\mu^1, \dots, \mu^{i-1}, \mu^{i+1}, \dots, \mu^n)$. A play $\rho : s_0 a_1 s_1 a_2 \dots a_k s_k$ is said to be consistent with a strategy μ^i if for all $j : 1 \leq j \leq k$, $a_j = \mu^i(s_j)$. It is easy to see that a strategy profile $\overline{\mu}$ defines a unique path in T, we denote this by $\rho_{\overline{\mu}}$.

Objectives. We consider simple reachability conditions for players. Each player is associated with a preference ordering $\preceq^i \subseteq$ *frontier*$(T) \times$ *frontier*(T) which is assumed to be a total pre-order over the leaf nodes of the tree. A game G is specified by the pair $G = (T, \{\preceq^i\}_{i \in N})$.

Nash equilibrium. Given a strategy profile μ^{-i}, we say μ^i is a best response for μ^{-i} if for all $\nu^i \in \Omega^i(T)$, *frontier*$(\rho_{(\nu^i, \mu^{-i})}) \preceq^i$ *frontier*$(\rho_{(\mu^i, \mu^{-i})})$. A strategy profile $\overline{\mu}$ constitutes a Nash equilibrium if for all $i \in N$, μ^i is a best response for μ^{-i}. A special case is when the preference ordering of players specify binary objectives, we refer to this as win-loss conditions for players. Two player zero sum games are ones in which $|N| = 2$ and the win-loss condition for each player is antagonistic. A strategy μ^i is said to be winning for player i if all plays consistent with μ^i satisfy the winning condition.

Extensive form games with imperfect information. Unlike perfect information games (as defined above), in a game with imperfect information, players might have uncertainty regarding the current game position. The standard way of modelling imperfect information is by associating an uncertainty relation $\sim^i \subseteq S \times S$ with each player $i \in N$. We assume that the uncertainty relation $\sim^i$ forms an equivalence relation and that it satisfies the condition of perfect recall which is defined as follows: for all s, $s' \in S$, and $a, a' \in \Sigma$ such that $a \in moves(s)$ and $a' \in moves(s')$, if $t \sim^i t'$ then $s \sim^i s'$ where $s \overset{a}{\Rightarrow} t$ and $s' \overset{a'}{\Rightarrow} t'$.

Since the uncertainty relation specifies game positions which players cannot distinguish between, we also require that the tree structure satisfies the condition: for all $i \in N$, for all $s, s' \in S^i$ if $s \sim^i s'$ then $moves(s) = moves(s')$. A strategy for player i is a function $\mu^i : S^i \to \Sigma$ which satisfies the condition that for all $s, s' \in S^i$, if $s \sim^i s'$ then $\mu^i(s) = \mu^i(s')$.

Non-determined games. Imperfect information games behave quite differently from their perfect information counterparts even at the level of two player zero sum games. It is well known that determinacy does not hold for the class of finite extensive form games with imperfect information as exhibited by the game of *matching pennies*. It is also easy to construct a non-zero sum variant of the game in which Nash equilibrium does not exist.

Question. In view of the above remark, it is quite natural to ask: given an imperfect information game G with win-loss objectives if it is decidable to check whether a winning strategy exists in G and for non-zero sum games whether a Nash equilibrium exists in G. For finite extensive form games it can be seen that both the questions are decidable essentially due to the fact that the strategy space is finite for all the players.

2.2 Unbounded Duration Games

In finite extensive form games, the duration of play is pre-determined in advance and it is commonly known to all players. More realistic game situations are ones in which the exact duration is not fixed but determined during the course of play. These are games of unbounded duration and can be modelled in terms of an infinite extensive form game. For algorithmic purposes, an interesting case is when the infinite tree is obtained as unfoldings of finite graphs (or game arenas).

In the case of game arenas, the strategy space of players is unbounded. Thus one cannot show decidability of the algorithmic questions posed earlier by simply enumerating all strategies. In fact it turns out that computing the winning strategy for the class of multiplayer games defined in terms of unfoldings of game arenas may be undecidable even for simple reachability objectives. Various versions of this result can be found in [14,3,2], borrowing elements from the seminal work of Peterson and Reif [13]. It is also easy to come up with a non-zero sum version where it can be shown that deciding the existence of Nash equilibrium is undecidable. The core idea in the undecidability results is to exploit the ability of the model to iteratively generate information forks [15]. Intuitively an information fork is a situation where two players have different information regarding the moves of a third player in a manner such that the uncertainty cannot be resolved locally.

This seems to suggest that models of imperfect information games which avoid information forks are better behaved in terms of algorithmic issues. We study a model arising from concurrency theory where uncertainty of players is structurally generated.

3 The Local Games Model

In the model we present below, each player is assumed to be making her moves locally, with the locations spatially distributed, and all information exchange being via communication. With each player $i \in N$, we associate a finite set Γ^i, the set of symbols which player i employs for communication. Let $\widetilde{\Gamma} = \Gamma^1 \times \cdots \times \Gamma^n$.

3.1 Game Arena

Local arena. For a player $i \in N$, the local game arena for i is a tuple $\mathcal{G}^i = (W^i, \rightarrow_i, w_0^i, \chi^i)$ where W^i is a finite set of local game positions, w_0^i is the initial game position, $\chi^i : W^i \rightarrow \Gamma^i$ associates with each local game position an element of Γ^i and the move function $\rightarrow_i : W^i \times \widetilde{\Gamma} \rightarrow 2^{W^i}$ satisfies the following condition: for all $w^i, v^i \in W^i$, if $w^i \xrightarrow{\gamma}_i v^i$ then $\gamma(i) = \chi^i(w^i)$.

The local game arena dictates the rules of the game for each player i. For each local game position w^i, the function χ^i specifies what player i communicates with the other players. The transition function takes into account the current game position and the communication received from other players to specify the set of possible moves enabled for player i. Communication in this model is by means of public announcements: for any state w^i, the value of $\chi^i(w^i)$ is communicated to all players, as we will see from the construction of the global arena below. A game structure $\mathbb{G}$ is defined in terms of a set of local game arenas for each player, $\mathbb{G} = \{\mathcal{G}^i\}_{i \in N}$.

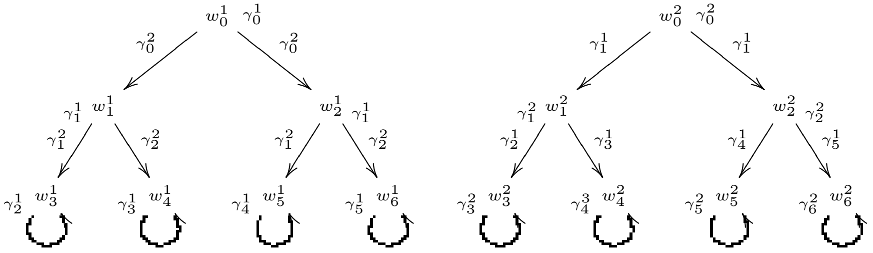
Global arena. Given a game structure $\mathbb{G} = \{\mathcal{G}^i\}_{i \in N}$, the resulting global game arena $\mathcal{G} = (W, \rightarrow, \mathbf{w}_0, \chi)$ is constructed as follows: the set of global game positions $W = W^1 \times \cdots \times W^n$ and $\mathbf{w}_0 = (w_0^1, \ldots, w_0^n)$. We define the function $\chi : W \rightarrow \widetilde{\Gamma}$ as $\chi(\mathbf{w}) = (\chi^1(w^1), \ldots, \chi^n(w^n))$ which associates with each global state, the announcements of players. The move relation $\rightarrow \subseteq W \times W$ satisfies the property: for all $\mathbf{w}, \mathbf{v} \in W$ we have $\mathbf{w} \rightarrow \mathbf{v}$ iff

- $\forall i \in \mathit{enabled}(\mathbf{w})$, $w^i \xrightarrow{\gamma}_i v^i$ where $\gamma = \chi(\mathbf{w})$.
- $\forall i \in N \setminus \mathit{enabled}(\mathbf{w})$, $v^i = w^i$.

where $\mathit{enabled}(\mathbf{w}) = \{i \in N \mid \exists v^i \in W^i \text{ with } w^i \xrightarrow{\gamma}_i v^i \text{ where } \gamma = \chi(\mathbf{w})\}$.

Plays. Since the global game arena is derived from the local arenas it is possible that there exist global game positions where moves of none of the players are enabled. In other words, these are game positions where the play is stuck and no progress can be made any further. For convenience, we think of such terminal game positions as sink nodes with a self loop. Thus a play in $\mathcal{G}$ is an infinite path $\rho = \mathbf{w}_0\mathbf{w}_1\ldots$ such that for all $j > 0$, we have $\mathbf{w}_{j-1}\rightarrow\mathbf{w}_j$. Let $states(\rho) = \{\mathbf{w} \in W \mid \exists j \text{ with } \mathbf{w} = \mathbf{w}_j\}$, i.e. it consists of all the global states occurring in ρ. For $i \in N$, let $states^i(\rho) = \{w^i \mid \mathbf{w} \in states(\rho)\}$. We denote the set of all plays in $\mathcal{G}$ by $Plays(\mathcal{G})$. We also use the notation $Paths(\mathcal{G})$ to denote the set of all finite partial plays in $\mathcal{G}$. For a partial play ϱ, we let $enabled(\varrho) = enabled(last(\varrho))$.

Views of players. For $i \in N$ and $\mathbf{w} \in W$, player i's view of $\mathbf{w}$ is defined as $view^i(\mathbf{w}) = (w^i, \chi(\mathbf{w}))$. For a sequence $\rho : \mathbf{w}_0\mathbf{w}_1\ldots$, we define player i's view of ρ as $view^i(\rho) = view^i(\mathbf{w}_0)view^i(\mathbf{w}_1)\ldots$. Let $Plays^i(\mathcal{G}) = \{view^i(\rho) \mid \rho \in Plays(\mathcal{G})\}$.



(a) $\mathcal{G}^1$

(b) $\mathcal{G}^2$

Fig. 1. Local game arenas

Example 3.1. Let the players be $N = \{1, 2\}$ and the communication alphabets be $\Gamma^1 = \{\gamma_0^1, \ldots, \gamma_5^1\}$ and $\Gamma^2 = \{\gamma_0^2, \ldots, \gamma_6^2\}$. Consider the local game arena $\mathcal{G}^1$ of player 1 given in Figure 1(a). The nodes of the graph corresponds to the local game positions, the announcements made by player 1 at each local state is marked along with the local states. For instance, $\chi^1(w_0^1) = \gamma_0^1$, $\chi^1(w_1^1) = \chi^1(w_2^1) = \gamma_1^1$ and so on. The self loop on states without any announcement annotation means that irrespective of the announcement made by the other player, the local state remains the same. The local arena $\mathcal{G}^2$ for player 2 is given in Figure 1(b). The derived global game graph is shown in Figure 2. In the global game graph, players 1 and 2 alternate moves till the game reaches one of the global sink nodes $\{(w_3^1, w_3^2), (w_4^1, w_4^2), (w_5^1, w_5^2), (w_6^1, w_6^2)\}$. Player 2 cannot distinguish between the global states (w_1^1, w_0^2) and (w_2^1, w_0^2) since $view^2((w_0^1, w_0^2)(w_1^1, w_0^2)) = view^2((w_0^1, w_0^2)(w_2^1, w_0^2))$.

The model does allow players to resolve imperfect information as the play progresses. For instance at the global state (w_3^1, w_3^2) player 2 knows that the

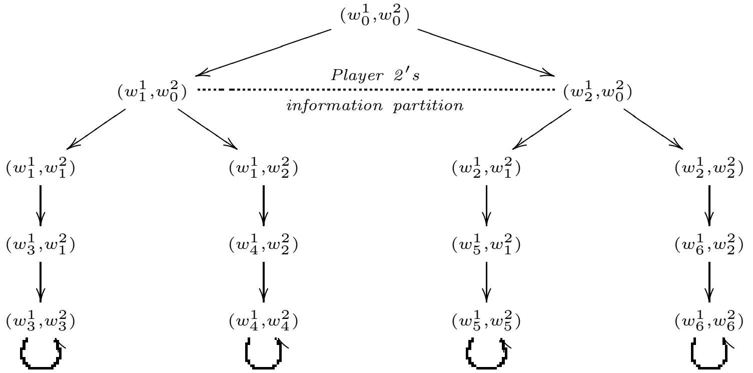


Fig. 2. Global game arena

play passed through the position (w_1^1, w_0^2) and not through (w_2^1, w_0^2). Thus the example demonstrates that "learning" is possible in the game model: basically, information sets of players can shrink during the course of play.

Note that the model is more general than turn based games where only one player can make a move at any given position. In the absence of communications there is no way of *scheduling* one player's move before another, and several players can move at the same position simultaneously. Communications can force turn based games also, as illustrated by Example 3.1.

In general, the communication alphabet of player i provides a projection of i's local states. The special case, when for all i, $\Gamma^i = W^i$ and χ_i is the identity map, gives us perfect information games, which is why we say that uncertainty in our model is generated structurally.

Extensive form tree. For an arena $\mathcal{G} = (W, \rightarrow, \mathbf{w}_0, \chi)$, the (infinite) extensive form game tree $T_\mathcal{G}$ associated with $\mathcal{G}$ is obtained by the tree unfolding of $\mathcal{G}$. In addition to keeping track of the sequence of game positions, we also keep track of the sequence of announcements made by players. Formally we have $T_\mathcal{G} = (S, \Rightarrow, s_0)$ where $S \subseteq (W \times \widetilde{\Gamma})^+$ and $\Rightarrow \subseteq S \times S$ are the least sets satisfying the condition: $(\mathbf{w}_0, \chi(\mathbf{w}_0)) \in S$ and

- If $s = (\mathbf{w}_0, \gamma_0) \dots (\mathbf{w}_k, \gamma_k) \in S$ and $\mathbf{w}_k \rightarrow \mathbf{w}'$ then $s' = s \cdot (\mathbf{w}', \chi(\mathbf{w}')) \in S$ and $s \Rightarrow s'$.

Since each node $s \in S$ corresponds to a path in $\mathcal{G}$, for each player $i \in N$, the notion of $view^i(s)$ is well defined. The uncertainty relation $\sim^i \subseteq S \times S$ is defined as follows: $s \sim^i s'$ iff $view^i(s) = view^i(s')$. It is easy to verify that $\sim^i$ thus defined forms an equivalence relation and that it satisfies perfect recall.

3.2 Strategies and Announcement Plans

Strategies. A strategy for player i, is a function $\mu^i : W^* \rightarrow W^i$ which satisfies the conditions: $\mu^i(\epsilon) = w_0^i$,

- for all finite partial plays $\varrho \in \mathit{Paths}(\mathcal{G})$, if $\mu^i(\varrho) = w^i$ then there exists $\mathbf{v}$ such that $\mathit{last}(\varrho) \rightarrow \mathbf{v}$ and $v^i = w^i$. This says that the strategy must choose only moves which are enabled.
- For all $\varrho, \varrho' \in \mathit{Paths}(\mathcal{G})$, if $\mathit{view}^i(\varrho) = \mathit{view}^i(\varrho')$ then $\mu^i(\varrho) = \mu^i(\varrho')$. This says that the strategy needs to respect the information partition.

We say that a play $\rho : \mathbf{w}_0\mathbf{w}_1 \ldots$ is consistent with strategy μ^i if $\forall j > 0$, $w^i_j = \mu^i(\mathit{view}^i(\mathbf{w}_0 \ldots \mathbf{w}_{j-1}))$. It is easy to see that a profile of strategies $\overline{\mu} = (\mu^1, \ldots, \mu^n)$ generates a unique path in the arena, we denote this by $\rho_{\overline{\mu}}$.

Announcement plans. An announcement plan for a player is similar to a strategy, except that instead of choosing a local state it chooses an observation. That is, an announcement plan is a map $\xi^i : W^* \rightarrow \Gamma^i$ which respects the information partition. This can also be thought of as a partially specified strategy, which instead of specifying an exact choice, restricts the available moves of the player. A strategy μ^i is said to conform to ξ^i if it satisfies the condition: for all partial plays ϱ, if $\mu^i(\varrho) = w^i$ then $\chi^i(w^i) = \xi^i(\varrho)$.

For an announcement plan ξ^i, let $[\![\xi^i]\!]$ denote the set of all strategies which conform to ξ^i. Given a strategy μ^i let $\mathfrak{p}(\mu^i)$ denote the announcement plan corresponding to μ^i. This is obtained by the simple transformation: for all $\varrho \in \mathit{Paths}(\mathcal{G})$, $\mathfrak{p}(\mu^i)(\varrho) = \chi^i(\mu^i(\varrho))$.

Advice automata. We are particularly interested in strategies which are represented by finite state machines. An advice automaton for player i over a game arena $\mathcal{G}$ is a structure $\mathcal{A}^i = (Q, \delta, o, q_0)$ consisting of a finite set of states Q, a transition function $\delta : Q \times W^i \times \widetilde{\Gamma} \rightarrow Q$, an advice function $o : Q \times W^i \times \widetilde{\Gamma} \rightarrow W^i$ along with an initial state $q_0 \in Q$.

A plan automaton $\mathcal{P}^i = (Q, \delta, o, q_0)$ for player i is similar to an advice automaton except for the advice function $o : Q \times W^i \times \widetilde{\Gamma} \rightarrow \Gamma^i$ which outputs an announcement instead of a local position.

3.3 Objectives

In this paper we restrict our attention to simple non-zero sum reachability objectives for players. For each player i, let $\mathcal{R}^i \subseteq W^i$. We assume that $\mathcal{R}^i$ constitutes sink nodes in the local arena, i.e. once the player reaches one of the states in $\mathcal{R}^i$ he stays in the state for ever. We also assume that for all $w^i \in \mathcal{R}^i$, $\chi^i(w^i) = w^i$. Let $\preceq^i \subseteq \mathcal{R}^i \times \mathcal{R}^i$ be a (local) preference ordering for each player. We assume that $\preceq^i$ forms a total pre-order over $\mathcal{R}^i$. A game is then specified by a pair $G = (\{\mathcal{G}^i\}_{i \in N}, \{\preceq^i\}_{i \in N})$.

We extend the preference relation over plays as follows: for $\rho, \rho' \in \mathit{Plays}(\mathcal{G})$, $\rho \preceq^i \rho'$ if $\mathit{states}^i(\rho) \cap \mathcal{R}^i \preceq^i \mathit{states}^i(\rho') \cap \mathcal{R}^i$. Note that for all $i \in N$ we assume the states in $\mathcal{R}^i$ are sink nodes. Thus in any play ρ, if the local state of player i hits a state in $\mathcal{R}^i$ then there will be a unique such state and therefore the above ordering over plays is well defined. We assume that plays which do not reach any of the reachability states are the least preferred ones. We find it useful to lift the preference ordering over paths to one over strategies for each player. For

$\mu^i, \nu^i \in \Omega(\mathcal{G})$, we define $\nu^i \trianglelefteq^i \mu^i$ if for all plays ρ_1 consistent with ν^i and for all plays ρ_2 consistent with μ^i, $\rho_1 \preceq^i \rho_2$.

We call a strategy μ^i dominant for player i in game G if for all $\nu^i \in \Omega^i(\mathcal{G})$, $\nu^i \trianglelefteq^i \mu^i$. The following proposition states that in the special case when G is a *perfect information* game, it is decidable to check if i has a dominant strategy.

Proposition 3.1. *Given a perfect information game G and a player i, it is decidable to check whether there exists a dominant strategy for player i and it is possible to synthesize a dominant strategy (when it exists).*

3.4 Locally Consistent Equilibrium

Local best response. Given a strategy profile μ^{-i}, a strategy μ^i of player i is a best response for μ^{-i} if for all $\nu^i \in \Omega^i(\mathcal{G})$, $\rho_{(\nu^i,\mu^{-i})} \preceq^i \rho_{(\mu^i,\mu^{-i})}$. For an announcement profile ξ^{-i}, we say that μ^i is a local best response for ξ^{-i} if for all $\nu^i \in \Omega^i(\mathcal{G})$ and for all $\mu^{-i} \in [\![\xi^{-i}]\!]$, $\rho_{(\nu^i,\mu^{-i})} \preceq^i \rho_{(\mu^i,\mu^{-i})}$. In other words, μ^i is a local best response if for all $\mu^{-i} \in [\![\xi^{-i}]\!]$, μ^i is the best response for μ^{-i}.

Equilibrium. A profile of strategies $\overline{\mu}$ constitutes a locally consistent equilibrium profile if for all $i \in N$, μ^i is a best response for $\mathfrak{p}(\mu^{-i})$.

How do locally consistent equilibria relate to the more standard notion of Nash equilibria? All solution concepts are justified by underlying notions of *rationality.* In our model, every player strategizes based on observations of announcements and the announcements she makes are strategic as well. No player has access to the epistemic structure of the other players, and hence offers a response not to any particular strategy of others, but to every potential strategy based on observations. In the case of perfect information games, when the communication alphabet is the same as the set of local states and announcement maps are identity functions, this notion is the same as Nash equilibrium.

Consider the game given in Example 3.1 (Figure 2). Suppose the preference ordering of player 1 is given by $w^1_4 \preceq^1 w^1_5 <^1 w^1_3 =^1 w^1_6$ and that of player 2 is $w^2_3 \preceq^2 w^2_6 <^2 w^2_4 =^2 w^2_5$. It is easy to check that the game does not have a Nash equilibrium, nor any locally consistent equilibrium.

Suppose that the preference orderings are as follows: $w^1_4 =^1 w^1_5 <^1 w^1_6 <^1 w^1_3$ and $w^2_4 =^2 w^2_5 <^2 w^2_3 <^2 w^2_6$. The game has two Nash equilibria. One in which player 1 chooses w^1_1 at the initial position and player 2 chooses w^2_1 subsequently. The second Nash equilibrium is the profile where player 1 chooses w^1_2 and player 2 chooses w^2_2. However note that choice of player 2 needs to be coordinated with that of player 1 to achieve equilibrium. Thus both the strategy profiles fail to be locally consistent equilibria. In fact it can be seen that this game does not have a locally consistent equilibrium.

On the other hand suppose the preference ordering is as follows: $w^1_6 <^1 w^1_4 <^1 w^1_5 =^1 w^1_3$ and $w^2_4 <^2 w^2_6 <^2 w^2_5 =^2 w^2_3$. Then there are two locally consistent equilibrium profiles in the game. One where player 1 chooses w^1_1 at the initial position and player 2 chooses w^2_1 subsequently and also the strategy profile where player 1 chooses w^1_2 followed by player 2 choosing w^2_1 subsequently.

Questions. For a game G the algorithmic questions of interest include:

- given an announcement profile ξ^{-i} in terms of bounded memory plan automata, is it possible to compute a local best response of player i for ξ^{-i} (when it exists)?
- is it decidable to check whether a locally consistent equilibrium profile exists in the game?
- is it possible to synthesize a locally consistent equilibrium profile (when it exists)?

4 Local Best Response Computation

Here we show that the local best response computation problem is decidable. Formally given a game $G = (\{\mathcal{G}^i\}_{i\in N}, \{\preceq^i\}_{i\in N})$, a bounded memory announcement profile ξ^{-i} in terms of plan automata, we show that it is possible to compute a local best response of player i (when it exists) for ξ^{-i}. The proof proceeds in two stages. For simplicity, for the rest of the section we assume that $i = 1$.

Restriction of the arena. Let $\mathcal{G} = (W, \rightarrow, w_0, \chi)$ and for all $j \in \{2, \dots, n\}$ let $\mathcal{P}^j = (Q^j, \delta^j, o^j, q_0^j)$ be the plan automaton corresponding to the bounded memory announcement plan ξ^j. We define the restriction of $\mathcal{G}$ with respect to the tuple $\mathcal{P} = (\mathcal{P}^2, \dots, \mathcal{P}^n)$ as follows: $\mathcal{G}_{\mathcal{P}} = (W_{\mathcal{P}}, \rightarrow_{\mathcal{P}}, \mathbf{w}_0^{\mathcal{P}}, \chi_{\mathcal{P}})$ where $W_{\mathcal{P}} = W \times Q^2 \times \ldots \times Q^n$, $\mathbf{w}_0^{\mathcal{P}} = (\mathbf{w}_0, \mathbf{q}_0)$ and $\chi_{\mathcal{P}}((\mathbf{w}, \mathbf{q})) = \chi(\mathbf{w})$. The edge relation satisfies the condition: $(\mathbf{w}_1, \mathbf{q}_1) \rightarrow_{\mathcal{P}} (\mathbf{w}_2, \mathbf{q}_2)$ iff $\mathbf{w}_1 \rightarrow \mathbf{w}_2$, for all $j \in \{2, \dots, n\}$ we have $o^j(q_1^j) = \chi^j(\mathbf{w}_2)$ and $\delta^j(q_1^j, w_1^j, \chi(\mathbf{w}_1)) = q_2^j$.

It can be verified that for all $j \in \{2, \dots, n\}$, $\Omega^j(\mathcal{G}_{\mathcal{P}})$ contains precisely those strategies in $\Omega^j(\mathcal{G})$ which conform to the announcement plan ξ^j.

Subset construction. Let $\mathcal{G}_{\mathcal{P}}$ be the restricted arena, we define the $\mathsf{knowledge\ arena}$ as follows: $\mathcal{G}^{\mathsf{K}} = (W^{\mathsf{K}}, \rightarrow^{\mathsf{K}}, w_0^{\mathsf{K}}, \chi^{\mathsf{K}})$ where $W^{\mathsf{K}} \subseteq 2^{W_{\mathcal{P}}}$ which satisfies the consistency condition that if $X \in W^{\mathsf{K}}$ then for all $(\mathbf{w}, \mathbf{q})$ and $(\mathbf{v}, \mathbf{q}') \in X$, $w^1 = v^1$. For a node $X \in W^{\mathsf{K}}$ and a state $u^j \in W^j$ we define $\mathsf{post}^{u^j}(X) = \{(\mathbf{w}, \mathbf{q}) \mid \exists (\mathbf{v}, \mathbf{q}') \text{ with } (\mathbf{w}, \mathbf{q}) \rightarrow_{\mathcal{P}} (\mathbf{v}, \mathbf{q}') \text{ and } v^j = u^j\}$. The move relation is defined as follows: $X \rightarrow^{\mathsf{K}} Y$ iff $\exists u^1$ such that $Y = \mathsf{post}^{u^1}(X)$. The initial position $w_0^{\mathsf{K}} = \{\mathbf{w}_0^{\mathcal{P}}\}$ and for $X \in W^{\mathsf{K}}$, $\chi^{\mathsf{K}}(X) = \chi^{\mathsf{K}}((\mathbf{w}, \mathbf{q}))$ for some $(\mathbf{w}, \mathbf{q}) \in X$. Due to the consistency requirement on W^{K} and the fact that the subset construction is performed on the restricted arena, the function χ^{K} is well defined. For $X \in W^{\mathsf{K}}$, we define $\mathit{view}^i(X) = (\mathit{view}^i(\mathbf{w}), \mathbf{q})$ for some $(\mathbf{w}, \mathbf{q}) \in X$.

Note that $\mathcal{G}^{\mathsf{K}}$ can be viewed as a *two player perfect information* game where players 2 to n can be thought of together as a second player playing against player 1. Thus in particular, due to Proposition 3.1 it is decidable to check whether player 1 has a dominant strategy in $\mathcal{G}^{\mathsf{K}}$.

Proposition 4.1. *Player 1 has a local best response for ξ^{-1} in $\mathcal{G}$ iff he has a dominant strategy in $\mathcal{G}^{\mathsf{K}}$.*

Proof. The crucial idea is to show the existence of a bijection $\mathfrak{b}$ between strategies in $\mathcal{G}^{\mathsf{K}}$ and those in $\mathcal{G}_{\mathcal{P}}$ which preserves outcome states in terms of the reachability condition. It can then be shown that if a dominant strategy μ^1_{K} exists in $\mathcal{G}^{\mathsf{K}}$ then the bounded memory strategy in $\mathcal{G}$ corresponding to $\mathfrak{b}(\mu^1_{\mathsf{K}})$ is a local best response for ξ^{-1}. If a dominant strategy does not exist in $\mathcal{G}^{\mathsf{K}}$ then by making use of the bijection $\mathfrak{b}$, it can be argued that there exist strategy profiles $\mu^{-1}_{\mathcal{P}}$ and $\nu^{-1}_{\mathcal{P}}$ for which player 1 does not have a common best response strategy. It then follows that player 1 does not have a local best response for ξ^{-1} in $\mathcal{G}$.

Corollary 4.1. *Given a game $G = (\{\mathcal{G}^i\}_{i\in N}, \{\preceq^i\}_{i\in N})$ and an announcement profile ξ^{-1}, it is decidable to check if player 1 has a local best response for ξ^{-1}. It is possible to synthesize a best response strategy when it exists.*

5 Equilibrium Computation

In this section we show that given a game G, it is decidable to check whether G has a locally consistent equilibrium profile. The main technique used is to construct a finite tree $T(\mathcal{G})$ from the arena $\mathcal{G}$ which preserves local outcomes of players. The tree $T(\mathcal{G})$ is constructed in such a manner that all strategies in $T(\mathcal{G})$ can be translated into equivalent bounded memory strategies in $\mathcal{G}$. We then employ a variant of the backward induction algorithm to synthesize a locally consistent equilibrium profile in $T(\mathcal{G})$ (if it exists) and show that the corresponding strategies in $\mathcal{G}$ form a locally consistent equilibrium profile in $\mathcal{G}$.

Let $G = (\{\mathcal{G}^i\}_{i\in N}, \{\preceq^i\}_{i\in N})$ and $\mathcal{G}$ be the corresponding global arena. For any partial play $\varrho = \mathbf{w}_0 \dots \mathbf{w}_k \in \mathit{Paths}(\mathcal{G})$ in which no state repeats and a state $\mathbf{w} \in W$, we define $\mathit{squeeze}(\varrho \cdot \mathbf{w})$ as follows: if $\exists j : 1 \le j \le k$ such that $\mathbf{w}_j = \mathbf{w}$ then $\mathit{squeeze}(\varrho \cdot \mathbf{w}) = \mathbf{w}_0 \dots \mathbf{w}_j$, otherwise $\mathit{squeeze}(\varrho \cdot \mathbf{w}) = \varrho \cdot \mathbf{w}$. Essentially the *squeeze* operator removes cycles from the path. This operation can be carried out (iteratively) for any play ρ. We sometimes abuse notation and simply refer to it as $\mathit{squeeze}(\rho)$. For a set $K \subseteq \mathit{Paths}(\mathcal{G})$, and observations $\gamma_1, \gamma_2 \in \widetilde{\Gamma}$, let $\mathit{update}(K, \gamma_1, \gamma_2) = \{\mathit{squeeze}(\varrho{\cdot}\mathbf{v}) | \varrho \in K, \forall i \in N, \mathit{last}(\varrho)^i \xrightarrow{\gamma_1}_i v^i$ and $\chi^i(v^i) = \gamma^i_2\}$.

The extensive form tree. For an arena $\mathcal{G}$, let $\mathcal{L}(\mathcal{G})$ denote the set of paths in $\mathcal{G}$ in which no game position repeats. Since $\mathcal{G}$ is finite, $\mathcal{L}(\mathcal{G})$ can be easily seen to be finite.

We define the extensive form tree as $T(\mathcal{G}) = (S, \Rightarrow, s_0, \mathfrak{l})$ where $S \subseteq W^+$ and $\mathfrak{l} : S \to 2^{\mathcal{L}(\mathcal{G})}$. Initially $S = \{\mathbf{w}_0\}$, $s_0 = \mathbf{w}_0$ and $\mathfrak{l}(s_0) = \{\mathbf{w}_0\}$. For any node $s \in S$, let ϱ_s denote the unique path from s_0 to s. We extend the tree by applying the following rule: pick any node $s \in S$ which satisfies the condition that there does not exist another node t in ϱ_s with $\mathfrak{l}(s) = \mathfrak{l}(t)$. Let $\mathit{last}(s) = \mathbf{w}$ and $\overrightarrow{\mathbf{w}}= \{\mathbf{w}_1, \dots, \mathbf{w}_k\}$. For each $j : 1 \le j \le k$ we add nodes $s \cdot \mathbf{w}_j$ to the set S. We extend the transition relation to include the edges $s{\Rightarrow}s \cdot \mathbf{w}_j$ and set $\mathfrak{l}(s \cdot \mathbf{w}_j) = \mathit{update}(\mathfrak{l}(s), \chi(\mathbf{w}), \chi(\mathbf{w}_j))$. The elements of $2^{\mathcal{L}(\mathcal{G})}$ represents the information the players need to keep track of during the course of play. We often refer to it as the knowledge set of players. At any node s, $K = \mathfrak{l}(s)$ provides the set of all (partial) plays that players consider possible given their sequence of observations. As noted earlier, the model allows the possibility of information sets of players to shrink as the play progresses. Thus on encountering a transition, the knowledge set is updated based on the new observable. This is done by eliminating all partial plays in K which are inconsistent with the new observation and extending the remaining plays with all states consistent with the observation.

Intuitively, the tree $T(\mathcal{G})$ is generated by the unfolding of $\mathcal{G}$ while also keeping track of the knowledge set associated with each node of the tree. The crucial idea is that since communication is by means of public announcement it suffices to keep track of a single set rather than a tuple of sets, one for each player. Observe that every play in $\mathcal{G}$ is carried over to $T(\mathcal{G})$ except that the play is terminated when the knowledge set repeats. Since the knowledge set has at most exponentially many elements in the number of permutations of positions in $\mathcal{G}$ and since every node in S has finite branching it follows that $T(\mathcal{G})$ is finite.

Uncertainty relation. We define the uncertainty relation for players over the nodes in $T(\mathcal{G})$ as follows: for all non-leaf nodes s, s' we have $s \approx^i s'$ iff $\mathit{last}(s)^i = \mathit{last}(s')^i$ and $\mathfrak{l}(s) = \mathfrak{l}(s')$. We assume that every leaf node is distinguishable by players and thus $s \approx^i s$ is the only relation that holds for a leaf state s.

Proposition 5.1. *For all non-leaf nodes $s, s' \in S$ and $i \in N$ if $s \approx^i s'$ then $|s| = |s'|$.*

This follows since for all non-leaf nodes $s \in S$, if $|s| = m$ then there exists a path $\rho \in \mathfrak{l}(s)$ such that $|\rho| = m$. An immediate consequence of Proposition 5.1 is that the uncertainty relation $\approx^i$ satisfies perfect recall.

Memoryless to bounded memory strategies. Given any strategy σ for player i in $T(\mathcal{G})$, we can construct a bounded memory strategy μ in $\mathcal{G}$ (denoted $\mathfrak{f}(\sigma)$) in terms of an advice automaton $\mathcal{A}^i$. The memory states of $\mathcal{A}^i$ are the subsets of $\mathcal{L}(\mathcal{G})$ (or the knowledge sets). The transition relation is the *update* operator defined on the knowledge sets. Note that the *update* operator is defined in terms of observables and not on the global states. Therefore this can be implemented by the transition relation of the advice automaton. The output function is simply the choice made by the strategy σ.

The only problem is with the leaf nodes of $T(\mathcal{G})$. Let s be a leaf node and s_1 be the node in ϱ_s such that $\mathfrak{l}(s) = \mathfrak{l}(s_1)$. Let $\mathit{last}(s) = w$ and $\mathit{last}(s_1) = w_1$. It is possible that $w^i \neq w_1^i$. However, for the strategy μ^i to be well defined, we need to show that μ^i specifies a consistent choice at $(w^i, \mathfrak{l}(s))$. Note that by construction, if $\mathfrak{l}(s) = \mathfrak{l}(s_1)$ then there exists a non-leaf node s_2 such that $|s_1| = |s_2|$ with $\mathit{last}(s_2)^i = w^i$ and $\mathfrak{l}(s_2) = \mathfrak{l}(s_1)$. Now since s_2 is a non-leaf node, by construction, a choice is specified for μ^i at $(w^i, \mathfrak{l}(s))$.

The backward induction procedure. As noted earlier, locally consistent equilibrium need not always exist. Starting at the leaf nodes, the backward induction procedure looks at every stage for the local best response of players. In case it does not exist, then the node is marked with $\downarrow$. For a tuple $\overline{x} = (x_1, \ldots, x_k)$ we say $\overline{x}$ is $\downarrow$-free if $\downarrow$ does not occur in $\overline{x}$. For each $i \in N$, we define a labelling function u^i on the frontier nodes of $T(\mathcal{G})$. For each frontier node s, let $\mathsf{u}^i(s) = (\mathit{last}(s))^i$. We denote by $\mathsf{u}(s)$ the tuple $(\mathsf{u}^1(s), \ldots, \mathsf{u}^n(s))$.

The backward induction procedure (given below) takes as input $T(\mathcal{G})$, the depth d of the tree $T(\mathcal{G})$ and involves d stages: starting at the leaf nodes and extending the labelling function to the interior nodes of $T(\mathcal{G})$.

Procedure $\mathsf{BI}(T(\mathcal{G}), \mathsf{d})$

- Initially, all the interior nodes of $T(\mathcal{G})$ are unlabelled.
- Repeat the following steps for $\mathfrak{c} = \mathsf{d} - 1$ to 1.
- For each $i \in N$ and information partition $\mathcal{I}^i$ of i at depth $\mathfrak{c}$ do the following:
    - Let $\mathcal{I}^i = \{s_1, \ldots, s_m\}$ and for each $k : 1 \leq k \leq m$ let $Y_k = \{\mathbf{v} \mid s_k \cdot \mathbf{v} \in S\}$. Let $Y_k^i = \{v^i \mid \mathbf{v} \in Y_k\}$, in other words Y_k^i consists of the moves of player i enabled at state s_k. Note that for all states in $\mathcal{I}^i$, the same set of moves are enabled for player i, let this set be $\{v_1^i, \ldots, v_r^i\}$. Let $X_{v_j^i} = \{\mathbf{v} \mid \exists s_k \in \mathcal{I}^i \text{ and } \mathit{last}(s_k) \rightarrow \mathbf{v} \text{ with } \mathbf{v}^i = \mathbf{v}_j^i\}$, this set is not empty since $v_j^i \in Y_k^i$. Let $\mathcal{L}^i(X_{v_j^i}) = \{\mathsf{u}^i(\mathbf{v}) \mid \mathbf{v} \in X_{v_j^i}\}$.
      Check if there exists a $X_{v_{\max}^i}$ such that $\mathcal{L}^i(X_{v_{\max}^i})$ is $\downarrow$-free and $\mathcal{L}^i(X_{v_j^i}) \preceq^i \mathcal{L}^i(X_{v_{\max}^i})$ for all j. If so then set $\sigma^i(s_k) = v_{\max}^i$ for all $s_k \in \mathcal{I}^i$ otherwise $\sigma^i(s_k)$ is not defined.
    - For a node s of depth $\mathfrak{c}$, for all $i \in \mathit{enabled}(s)$ if $\sigma^i(s)$ is defined then set $\mathsf{u}^i(s) = \mathsf{u}^i(\sigma(s))$ else $\mathsf{u}^i(s) = \downarrow$.

Let $\approx^*$ denote the transitive closure of $\{(s, s') \in S \times S \mid s \approx^i s' \text{ for some } i \in N\}$. Let $Z_s = \{s' \mid s \approx^* s'\}$. Let $T(Z_s)$ denote the forest consisting of all the subtrees rooted at nodes in Z_s and let $T(\mathcal{G}, s, \approx^*)$ denote the tree where we add a unique root r_0 along with edges to all the nodes in Z_s. The moves of players at r_0 is defined in the obvious manner. The following lemma asserts the correctness of the backward induction procedure.

Lemma 5.1. *For a tree $T(\mathcal{G})$, for all $s \in S$ there exists a locally consistent equilibrium in $T(\mathcal{G}, s, \approx^*)$ with outcome $(w^1, \ldots, w^n)$ iff $\mathsf{u}(s) = (w^1, \ldots, w^n)$ and $\downarrow$-free.*

Proof. The proof is by induction on $|s|$. The base case is when $|s| = \mathsf{d}$ which corresponds to the leaf nodes. In this case the only uncertainty for players is $s \approx^i s$ and therefore the lemma follows easily. Suppose the claim holds for all s with $|s| < \mathfrak{c}$ and let $s \in S$ with $|s| = \mathfrak{c}$. Suppose $\mathsf{u}(s)$ is $\downarrow$-free and $\mathsf{u}(s) = (w^1, \ldots, w^n)$. According to step 2 of the backward induction procedure, for all $i \in \mathit{enabled}(s)$, $\sigma^i(s)$ is defined and $\mathsf{u}^i(\sigma^i(s)) = w^i$. If $\sigma^i(s)$ is defined then it means that for all $i \in \mathit{enabled}(s)$, there exists a $v_{\max}^i$ such that $\mathcal{L}^i(X_{v_j^i}) \preceq^i \mathcal{L}^i(X_{v_{\max}^i})$ for all j and $\sigma^i(s) = v_{\max}^i = \mathbf{w}^i$. By induction hypothesis, for all $s' \in X_{v_j^i}$, there exists a locally consistent equilibrium in $T(\mathcal{G}, s', \approx^*)$ with outcome $\mathsf{u}^i(s') = w^i$.

Since $\mathcal{L}^i(X_{v^i_j}) \preceq^i \mathcal{L}^i(X_{v^i_{\max}})$ for all j we have $\sigma^i(s) = v^i_{\max}$ constitutes a locally consistent equilibrium in $T(\mathcal{G}, s, \approx^*)$.

Suppose there exists a locally consistent equilibrium in $T(\mathcal{G}, s, \approx^*)$ with the outcome $(w^1, \ldots, w^n)$. Then for all $i \in \mathit{enabled}(s)$, for all $\mathcal{I}^i \subseteq Z_s$, there exists a $v^i_{\max}$ such that $s' = s \cdot \mathbf{v}_{\max}$ and there exists a locally consistent equilibrium in $T(\mathcal{G}, s', \approx^*)$ with outcome $(w^1, \ldots, w^n)$. By induction hypothesis, $\mathsf{u}(s') = (w^1, \ldots, w^n)$. By the backward induction procedure, we get $\mathsf{u}(s) = (w^1, \ldots, w^n)$ and $\sigma^i(s) = v^i_{\max}$.

Lemma 5.2. *For a game $G = (\{\mathcal{G}^i\}_{i \in N}, \{\preceq^i\}_{i \in N})$, a locally consistent equilibrium profile exists in $\mathcal{G}$ iff a locally consistent equilibrium profile exists in $T(\mathcal{G})$.*

Proof. It can be shown that if a locally consistent equilibrium profile $\overline{\sigma}$ exists in $T(\mathcal{G})$ then $\overline{\mu} = \mathfrak{f}(\overline{\sigma})$ constitutes an equilibrium profile in $\mathcal{G}$. Conversely, if there does not exist a locally consistent equilibrium in $T(\mathcal{G})$ then at each level of the tree we get witness information sets which violate the local consistency requirement of equilibrium based on the observations of players. This essentially means the existence of a player for which a common best response strategy for any play conceived possible by his observations does not exist. These witness sets can be lifted to paths in $\mathcal{G}$ as specified by the knowledge set.

Theorem 5.1. *Given a game $G = (\{\mathcal{G}^i\}_{i \in N}, \{\preceq^i\}_{i \in N})$, it is decidable to check whether a locally consistent equilibrium profile exists in G. It is possible to synthesize such an equilibrium profile (when it exists).*

Proof. From Lemma 5.2 it follows that to decide if a locally consistent equilibrium profile exists in G we can run the backward induction algorithm on $T(\mathcal{G})$ and check if the labelling at the root is ↓-free. By Lemma 5.1, the strategy profile $\overline{\sigma}$ constructed is a locally consistent equilibrium in $T(\mathcal{G})$ and from Lemma 5.2 we have that $\mathfrak{f}(\overline{\sigma})$ is a locally consistent equilibrium profile in G.

6 Discussion

In the model discussed here communication is by means of public announcement, but the model is capable of dealing with other forms of communication. For instance, private communication can be captured by associating with each pair of players a set of announcement alphabets which is then communicated to the specific player by means of a private channel. It can be shown that in this setting the question of whether Nash equilibrium exists is undecidable. Public and private communication form two extremes and ongoing work includes looking at the notion of locally consistent equilibrium for other types of restricted communication models.

Acknowledgements. We thank the anonymous referees for their valuable comments and suggestions. The first author thanks the Netherlands Institute for Advanced Study in the Humanities and Social Sciences for its support. The second author acknowledges the support of the European Commission grant EMECW15.

References

1. Arnold, A., Walukiewicz, I.: Nondeterministic controllers of nondeterministic processes. In: Logic and Automata: History and Perspectives. Texts in Logic and Games, vol. 2, pp. 29–52. Amsterdam University Press (2007)
2. Bernet, J.: Discrete games for the synthesis and validation of communication processes. PhD thesis, University of Bordeaux (2006)
3. Bernet, J., Janin, D.: Automata theory for discrete distributed games. Technical report, LaBRI, University of Bordeaux (2005)
4. Berwanger, D., Kaiser, L.: Information tracking in games on graphs. Journal of Logic, Language and Information (2010)
5. Chatterjee, K., Doyen, L., Henzinger, T., Raskin, J.-F.: Algorithms for omega-regular games of imperfect information. Logical Methods in Computer Science 3(3:4) (2007)
6. Gastin, P., Lerman, B., Zeitoun, M.: Distributed games and distributed control for asynchronous systems. In: Farach-Colton, M. (ed.) LATIN 2004. LNCS, vol. 2976, pp. 455–465. Springer, Heidelberg (2004)
7. Gastin, P., Sznajder, N., Zeitoun, M.: Distributed synthesis for well-connected architectures. Formal Methods in System Design 34, 215–237 (2009)
8. Gripon, V., Serre, O.: Qualitative concurrent stochastic games with imperfect information. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 200–211. Springer, Heidelberg (2009)
9. Halpern, J.: A computer scientist looks at game theory. Games and Economic Behavior 45(1) (2003)
10. Hintikka, J., Sandu, G.: Informational independence as a semantical phenomenon. In: Logic, Methodology and Philosophy of Science, vol. 8, pp. 571–589. Elsevier, Amsterdam (1989)
11. Lamport, L., Pease, M., Shostak, R.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems 4(3), 382–401 (1982)
12. Mohalik, S., Walukiewicz, I.: Distributed games. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 338–351. Springer, Heidelberg (2003)
13. Peterson, G., Reif, J.: Multi-person alternation. In: Proceedings of FOCS 1979, pp. 348–363. IEEE, Los Alamitos (1979)
14. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proceedings of FOCS 1990, pp. 746–757 (1990)
15. Schewe, S.: Uniform distributed synthesis. In: Proceedings of LICS 2005, pp. 321–330 (2005)

Flat Coalgebraic Fixed Point Logics

Lutz Schröder[1,⋆] and Yde Venema[2]

[1] DFKI Bremen and Department of Computer Science, Universität Bremen
[2] ILLC, Universiteit van Amsterdam

Abstract. Fixed point logics are widely used in computer science, in particular in artificial intelligence and concurrency. The most expressive logics of this type are the μ-calculus and its relatives. However, popular fixed point logics tend to trade expressivity for simplicity and readability, and in fact often live within the single variable fragment of the μ-calculus. The family of such *flat* fixed point logics includes, e.g., CTL, the $*$-nesting-free fragment of PDL, and the logic of common knowledge. Here, we extend this notion to the generic semantic framework of *coalgebraic logic*, thus covering a wide range of logics beyond the standard μ-calculus including, e.g., flat fragments of the graded μ-calculus and the alternating-time μ-calculus (such as ATL), as well as probabilistic and monotone fixed point logics. Our main results are completeness of the Kozen-Park axiomatization and a timed-out tableaux method that matches EXPTIME upper bounds inherited from the coalgebraic μ-calculus but avoids using automata.

1 Introduction

Many of the most well-known logics in program verification, concurrency, and other areas of computer science and artificial intelligence can be cast as fixed point logics, that is, embedded into some variant of the μ-calculus. Typical examples are PDL [25] where, say, the formula $\langle a^*\rangle p$ ('p can be reached by finite iteration of a') can be expressed as the least fixed point $\mu X.\, p\vee\langle a\rangle X$; CTL [7], whose formula AFp ('p eventually holds on all paths') is just the fixed point $\mu X.\, p\vee\Box X$; and the common knowledge operator C of epistemic logic [19], where Cp ('it is common knowledge that p') can be expressed as the fixed point $\nu X.\, \bigwedge_{i=1}^{n} K_i(p\wedge X)$ with n the number of agents and K_i read as 'agent i knows that'. A common feature of these examples is that they trade off expressivity for simplicity of expression in comparison to the full μ-calculus.

One of the reasons why the full μ-calculus is both hard to read and algorithmically problematic in practice is that one has to keep track of bound variables. Indeed we note that the simpler logics listed above (in the case of PDL, the $*$-nesting-free fragment) live in the single-variable fragment of the μ-calculus (a subfragment of the alternation-free fragment [10]), which is precisely what enables one to abandon variables altogether in favour of variable-free fixed point operators such as AF or C. We refer to logics that embed into a single-variable μ-calculus as *flat fixed point logics* [27].

Here, we study flat fixed point logics in the more general setting of *coalgebraic logic*. Coalgebra has recently emerged as the right framework for a unified treatment of a wide range of modal logics with seemingly disparate semantics beyond the realm of

[⋆] Work performed as part of the project FormalSafe funded by the German Federal Ministry of Education and Research (FKZ 01IW07002).

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 524–538, 2010.

pure relational structures. Examples include monotone modal logic, probabilistic modal logics [17], graded modal logic [11,5], and coalition logic [23]. This level of generality is achieved by parametrizing the semantics over a type functor on the category of sets, whose coalgebras play the role of frames. Besides standard Kripke frames, the notion of coalgebra encompasses, e.g., Markov chains, weighted automata, multigraphs, neighbourhood frames, selection function frames, and concurrent game structures. The theory of coalgebraic modal logic has evolved quite rapidly, and presently includes, e.g., generic upper bounds PSPACE for satisfiability in next-step logics [29], and EXPTIME for satisfiability under global assumptions in hybrid next-step logics [31].

In our *flat coalgebraic fixed point logics* one can express operators such as 'the coalition C of agents can maintain p forever', 'the present state is the root of a binary tree all whose leaves satisfy p', or 'p is commonly believed with reasonable certainty'. In particular, we cover the single-variable fragments of the graded μ-calculus [16] and the alternating-time μ-calculus (AMC) [1], including alternating-time temporal logic (ATL). Flat coalgebraic fixed point logics are fragments of coalgebraic μ-calculi, and as such known to be decidable in EXPTIME under reasonable assumptions [4]. However, the decision procedure for the coalgebraic μ-calculus is, like the one for the standard μ-calculus [9], based on automata and as such has exponential average-case run time, while tableaux methods as suggested, e.g., by Emerson and Halpern for CTL [8] and by Kozen for the aconjunctive fragment of the μ-calculus [14] are expected to offer the possibility of feasible average-case behaviour.

Our main results on flat coalgebraic fixed point logics, parametric both w.r.t. the coalgebraic branching type and the choice of flat fragment, are

- completeness of the natural axiomatization that makes the fixed point definitions explicit, generalizing the well-known Kozen-Park axiomatization; and
- a construction of *timed-out tableaux* similar in spirit to Kozen's tableaux for the aconjunctive μ-calculus,

both under mild restrictions on the form of fixed point operators. The completeness result generalizes results of [27] to the level of coalgebraic logic, and relies on the notion of *$\mathcal{O}$-adjointness* [26] to prove that fixed points *in the Lindenbaum algebra* are *constructive*, i.e. approximable in ω steps. The crucial ingredient here are the *one-step cutfree complete* rule sets of [29,22]. These enable significant generalizations of both the key *rigidity lemma* and the $\mathcal{O}$-adjointness theorem of [27], the latter to the effect that *all uniform-depth modal operators are $\mathcal{O}$-adjoint*. The novel tableaux construction is instrumental in the completeness proof, and at the same time confirms the known EXPTIME upper bound, avoiding however the use of automata and thus raising hopes for efficient implementation.

Our completeness result follows a long tradition of non-trivial completeness proofs, e.g. for PDL [15,32], CTL [8], the aconjunctive μ-calculus [14], and the full μ-calculus [33]. Note that all these results are independent, as completeness is not in general inherited by sublogics, and in fact employ quite different methods. Instantiating our generic results to concrete logics yields new results in nearly all cases that go beyond the classical relational μ-calculus, noting that neither [16] nor [4] cover axiomatizations. In particular, we obtain for the first time a completeness result and a tableau procedure for graded fixed point logics, i.e. fragments of the graded μ-calculus, and we generalize the completeness of ATL [12] to arbitrary flat fragments of AMC.

2 Flat Coalgebraic Fixed Point Logics

We briefly recall the generic framework of coalgebraic modal logic [21,28], and define its extension with flat fixed point operators, a fragment of the coalgebraic μ-calculus [4].

The first parameter of the syntax is a *(modal) similarity type* Λ, i.e. a set of modal operators with associated finite arities. We work with formulas in negation normal form throughout, and therefore assume that every modal operator $\heartsuit \in \Lambda$ comes with a *dual* operator $\overline{\heartsuit} \in \Lambda$ of the same arity, where $\overline{\overline{\heartsuit}} = \heartsuit$. This determines the set $\mathcal{F}(\Lambda)$, or just $\mathcal{F}$, of *modal formulas* γ, δ by the grammar

$$\gamma, \delta ::= \bot \mid \top \mid v \mid \neg v \mid \gamma \wedge \delta \mid \gamma \vee \delta \mid \heartsuit(\gamma_1, \dots, \gamma_n)$$

where $\heartsuit \in \Lambda$ is n-ary and $v \in V$ for a fixed countably infinite set V of *variables*. Negation $\neg$, admitted in the above grammar only for variables v, then becomes a derived operation on all formulas in the standard way; e.g., $\neg\heartsuit(\gamma_1, \dots, \gamma_n) = \overline{\heartsuit}(\neg\gamma, \dots, \neg\gamma_n)$, and $\neg\neg v = v$. Further derived operations $\rightarrow$, $\leftrightarrow$ are defined as usual. Moreover, we define the *dual* $\overline{\gamma}$ of γ as $\overline{\gamma} \equiv \neg\gamma\sigma$ where the substitution σ is given by $\sigma(v) = \neg v$ for all $v \in V$. We intend variables as place holders for arguments and parameters of formulas defining fixed point operators; as such, they serve only technical purposes and will not form part of the actual fixed point language defined below. Instead, propositional atoms are incorporated into the modal similarity type Λ as nullary operators when needed.

The second syntactic parameter of a *flat coalgebraic fixed point logic* is a set Γ of modal formulas γ, where we distinguish a single fixed *argument* variable x and regard all other variables $p_1, \dots, p_n$ in γ as *parameters*; we require that γ is *monotone* in all variables, i.e. does not contain $\neg x$ (an essential condition for the existence of fixed points) or $\neg p_i$ (a mere technical convenience, and not an actual restriction as one can always negate the actual parameter instead of the parameter variable). We require moreover that all $\gamma \in \Gamma$ are *guarded*, i.e. that all occurrences of the argument variable x are under the scope of at least one modal operator; as shown in [33], this is not an essential restriction as every μ-calculus formula is provably equivalent to a guarded formula. We denote substituted formulas $\gamma[\varphi_1/p_1; \dots; \varphi_n/p_n; \psi/x]$ as $\gamma(\varphi_1, \dots, \varphi_n, \psi)$. The set $\mathcal{F}_\sharp(\Lambda, \Gamma)$ or just $\mathcal{F}_\sharp$ of *(fixed point) formulas* φ, ψ is then defined by the grammar

$$\varphi, \psi ::= \bot \mid \top \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \heartsuit(\varphi_1, \dots, \varphi_n) \mid \sharp_\gamma(\varphi_1, \dots, \varphi_n) \mid \flat_{\overline{\gamma}}(\varphi_1, \dots, \varphi_n)$$

where $\heartsuit \in \Lambda$ is n-ary and $\gamma \in \Gamma$. The operator $\sharp_\gamma$ represents the least fixed point

$$\sharp_\gamma(\varphi_1, \dots, \varphi_n) = \mu x.\gamma(\varphi_1, \dots, \varphi_n, x),$$

while $\flat_{\overline{\gamma}}(\varphi_1, \dots, \varphi_n)$ represents the greatest fixed point $\nu x.\overline{\gamma}(\varphi_1, \dots, \varphi_n, x)$. The name *flat* for the fixed point operators $\sharp_\gamma$, $\flat_{\overline{\gamma}}$ relates to the fact that we require the formula γ to belong to the basic (fixed point free) modal language. Note that nesting of fixed point operators is unrestricted, e.g. φ can be an arbitrary fixed point formula in $\sharp_\gamma\varphi$. Syntactically, $\sharp_\gamma$ is an atomic operator, and occurrences of variables in γ do not count as occurrences in formulas $\sharp_\gamma\phi$. For the sake of readability, we restrict the further technical development (but not the examples) to unary modalities $\heartsuit$ and unary fixed point operators, i.e. we assume that every $\gamma \in \Gamma$ has only one parameter variable, *denoted by p throughout*; the extension to higher arities is a mere notational issue. Negation extends to fixed points by $\neg\sharp_\gamma\varphi = \flat_{\overline{\gamma}}(\neg\varphi)$ and $\neg\flat_{\overline{\gamma}}\varphi = \sharp_\gamma(\neg\varphi)$. Note that unlike in the case of modal formulas, we have not included variables in the definition of fixed point formulas. A *(fixed point) formula with variables* is an expression of the form $\gamma\sigma$, where γ is a modal formula and σ is a substitution of some of the variables in γ with fixed point formulas (i.e. variables never appear under fixed point operators). In the following, the term *formula* will refer to fixed point formulas without variables unless variables are explicitly mentioned. For $\gamma \in \Gamma$, we denote the function taking a formula ψ to $\gamma(\varphi, \psi)$ by $\gamma(\varphi)$, and by $\gamma(\varphi)^k$ its k-fold iteration. *We assume a reasonable size measure on Λ* and hence on formulas and sets of formulas [30], in particular that numbers (e.g. in graded or probabilistic operators) are *coded in binary*.

The logic is further parametrized *semantically* over the underlying class of systems and the interpretation of the modal operators. The former is determined by the choice of a *type functor* $T : \mathsf{Set} \to \mathsf{Set}$, i.e. an operation T that maps sets X to sets TX and functions $f : X \to Y$ to functions $Tf : TX \to TY$, preserving identities and composition, and the latter by the choice of a predicate lifting $[\![\heartsuit]\!]$ for each $\heartsuit \in \Lambda$. Here, a *predicate lifting* (for T) is a family of maps $\lambda_X : \mathcal{P}X \to \mathcal{P}TX$, where X ranges over all sets, satisfying the *naturality* condition $\lambda_X(f^{-1}[A]) = (Tf)^{-1}[\lambda_Y(A)]$ for all $f : X \to Y$, $A \in \mathcal{P}Y$. As we work with fixed points, we insist that *all modal operators are monotone*, i.e. $[\![\heartsuit]\!] : \mathcal{P}(X) \to \mathcal{P}(TX)$ is monotone w.r.t. set inclusion for each $\heartsuit \in \Lambda$. Moreover, the assignment of predicate liftings must respect duality of operators: for $\heartsuit \in \Lambda$, $[\![\overline{\heartsuit}]\!]_X(A) = TX - [\![\heartsuit]\!]_X(X - A)$. Given these data, the role of models is played by T*-coalgebras*, i.e. pairs (X, ξ) where X is a set of *states* and $\xi : X \to TX$ is the transition function; thinking of TX informally as a parametrised datatype over X, we regard ξ as associating with each state x a structured collection $\xi(x)$ of successor states and observations. E.g. for $TX = \mathcal{P}X \times \mathcal{P}(U)$, given a set U of propositional atoms, we obtain that T-coalgebras are Kripke models, as they associate with each state a set of successor states and a set of valid propositional atoms. Our main interest here is in examples beyond Kripke semantics, see Example 1.

As indicated above, the choice of predicate liftings determine the interpretation of modal operators. The semantics of a formula φ with argument variable x (no other variables will ever be evaluated in unsubstituted form) is a subset $[\![\varphi]\!]_{(X,\xi)}(B) \subseteq X$, given a T-coalgebra (X, ξ) and a set $B \subseteq X$. The semantics of formulas φ without variables (in particular of $\sharp$- or $\flat$-formulas) does not depend on B and hence will be denoted just by $[\![\varphi]\!]_{(X,\xi)}$. One has obvious clauses for Boolean operators, $[\![x]\!]_{(X,\xi)}(B) = B$, and

$$\begin{aligned}
[\![\heartsuit\varphi]\!]_{(X,\xi)}(B) &= \xi^{-1}[\![\heartsuit]\!]_X([\![\varphi]\!]_{(X,\xi)}(B)) \\
[\![\sharp_\gamma\varphi]\!]_{(X,\xi)} &= \bigcap\{B \subseteq X \mid [\![\gamma(\varphi)]\!]_{(X,\xi)}(B) \subseteq B\} \\
[\![\flat_{\overline{\gamma}}\varphi]\!]_{(X,\xi)} &= \bigcup\{B \subseteq X \mid B \subseteq [\![\overline{\gamma}(\varphi)]\!]_{(X,\xi)}(B)\}.
\end{aligned}$$

The clause for $\sharp_\gamma\varphi$ just says that $[\![\sharp_\gamma\varphi]\!]_{(X,\xi)}$ is the least fixed point of the monotone map $[\![\gamma(\varphi)]\!]_{(X,\xi)} : \mathcal{P}(X) \to \mathcal{P}(X)$, and similarly $[\![\flat_{\overline{\gamma}}\varphi]\!]_{(X,\xi)}$ is the greatest fixed point of $[\![\overline{\gamma}(\varphi)]\!]_{(X,\xi)}$. *We fix the data T, Λ, Γ etc. throughout.*

Example 1. 1. *Kripke semantics:* Fixed point extensions of the modal logic K have a single modal operator $\Box$, interpreted over the powerset functor $\mathcal{P}$ (which takes a set X to its powerset $\mathcal{P}(X)$) by the predicate lifting $[\![\Box]\!]_X(A) = \{B \in \mathcal{P}(X) \mid B \subseteq A\}$. $\mathcal{P}$-coalgebras $(X, \xi : X \to \mathcal{P}(X))$ are in 1-1 correspondence with Kripke frames, and $[\![\Box]\!]$ captures the usual semantics of the box operator. Multi-agent extensions are interpreted over $TX = \mathcal{P}(A \times X)$ where A is the set of agents. CTL, $*$-nesting-free PDL, and the logic of common knowledge are flat fixed point logics in this setting; e.g, AU and EU are the $\sharp$-operators for $p_2 \vee (p_1 \wedge \Box x)$ and for $p_2 \vee (p_1 \wedge \Diamond x)$, respectively.

2. *Graded fixed point logics* are sublogics of the *graded μ-calculus* [16]. They have modal operators $\Diamond_k$ 'in more than k successors', with duals $\Box_k$ 'in all but k successors', interpreted over the functor $\mathcal{B}$ that takes a set X to the set $\mathcal{B}(X) = X \to \omega + 1$ of multisets over X by $[\![\Diamond_k]\!]_X(A) = \{B \in \mathcal{B}(X) \mid \sum_{x \in A} B(x) > k\}$. This captures the semantics of graded modalities over *multigraphs* [5], which is equivalent to the more customary Kripke semantics [11] w.r.t. satisfiability of fixed point formulas. In description logic, graded operators are called *qualified number restrictions* [2]. The example mentioned in [16], a graded fixed point formula expressing that the current state is the root of a finite binary tree all whose leaves satisfy p, can be expressed by the $\sharp$-operator for $p \vee \Diamond_1 x$. Similarly, the $\sharp$-operator for $p \vee \Box_k x$ expresses that p holds somewhere on every infinite $k+1$-ary tree starting at the current state.

3. *Probabilistic fixed point logics*, i.e. fixed point extensions of probabilistic modal logic [17], have modal operators L_p 'in the next step, it holds with probability at least p that', for $p \in [0,1] \cap \mathbb{Q}$. They are interpreted over the functor $\mathcal{D}$ that maps a set X to the set of discrete probability distributions on X by putting $[\![L_p]\!]_X(A) = \{P \in \mathcal{D}(X) \mid PA \geq p\}$. Coalgebras for $\mathcal{D}$ are Markov chains. We can use the $\flat$-operator AG_p for $p \wedge L_p x$ to express formulas like $AG_p \neg \mathsf{fail}$, stating that the system will, at any point during its run time, fail with probability at most $1-p$; a sensible specification for systems that may sometimes fail but should not fail excessively often. In an epistemic reading of probabilities, flat probabilistic fixed point logics support, e.g., a common belief operator 'it is commonly believed with confidence p that'.

4. *The alternating-time μ-calculus (AMC)* [1] has modal operators $\langle\!\langle A \rangle\!\rangle \bigcirc$ read 'coalition A has a joint strategy to enforce ... in one step', where a *coalition* is a subset of a fixed set N of agents (in coalition logic [23], these operators are denoted $[A]$). Their semantics is defined over *concurrent game structures* (or *game frames*), and can be captured coalgebraically [29]. One of the flat fragments of AMC is Alternating-Time Temporal Logic (ATL) [1]. E.g., the ATL-operator $\langle\!\langle A \rangle\!\rangle p_1 \mathcal{U} p_2$, read 'coalition A can eventually force p_2 and meanwhile maintain p_1', is the $\sharp$-operator for $p_2 \vee (p_1 \wedge \langle\!\langle A \rangle\!\rangle \bigcirc x)$. Flat fixed points in AMC go considerably beyond ATL; e.g. the $\flat$-operator for $p \wedge \langle\!\langle\rangle\!\rangle \bigcirc \langle\!\langle\rangle\!\rangle \bigcirc x$ ('p holds in all even states along any path') is not even in ATL* [1,6]. A similar flat operator, the $\flat$-operator for $\langle\!\langle A \rangle\!\rangle \bigcirc (p \wedge \langle\!\langle B \rangle\!\rangle \bigcirc (q \wedge x))$, expresses that coalitions A and B can forever play ping-pong between p and q.

5. *Monotone fixed point logics* have a modal operator $\Box$, interpreted over the *monotone neighbourhood functor* defined by $\mathcal{M}(X) = \{\mathfrak{A} \in \mathcal{P}(\mathcal{P}(X)) \mid \mathfrak{A} \text{ upwards closed}\}$ by means of the predicate lifting $[\![\Box]\!]_X(A) = \{\mathfrak{A} \in \mathcal{M}(X) \mid A \in \mathfrak{A}\}$. In multi-modal versions of this, boxes and their semantics are indexed, e.g. over agents, programs, or games. This is the semantic setting of logics such as concurrent

PDL [24] and Parikh's game logic [20], the flat fragments of which are the $*$-nesting-free fragments. E.g., using $\langle\gamma\rangle$ to denote the game logic operator 'Angel has a strategy to enforce ... in game γ', the operator $\langle\gamma^\times\rangle$ for a $*$-free game γ, where $\times$ denotes demonic iteration (Demon chooses the number of rounds), is the $\flat$-operator for $p \wedge \langle\gamma\rangle x$.

3 The Generic Axiomatization

The generic semantic and syntactic framework of the previous section comes with a generic, parametrized *deduction system*, whose completeness will be one of our main results. We begin with the fixed part of the deduction system. We include full *propositional reasoning*, i.e. introduction of substituted propositional tautologies and modus ponens. Fixed points are governed by the obvious generalization of the *Kozen-Park axiomatization*: we have the *unfolding* axiom

$$\sharp_\gamma\varphi \leftrightarrow \gamma(\varphi, \sharp_\gamma\varphi)$$

and the *fixed-point induction* rule

$$\gamma(\varphi, \chi) \to \chi \;/\; \sharp_\gamma\varphi \to \chi,$$

for all formulas φ, χ. (Here $\alpha \;/\; \beta$ denotes the rule 'from α infer β').

The variable part is now the axiomatization of the modal operators, which turns out to be completely orthogonal to the fixed point axiomatization. In fact, we can just re-use complete rule sets for the purely modal logic as developed in [29]. First some notation.

Definition 2. We denote the set of of positive propositional formulas (formed using only $\wedge$ and $\vee$) over a set Z by $\mathsf{Pos}(Z)$, and the set $\{\heartsuit a \mid \heartsuit \in \Lambda, a \in Z\}$ by $\Lambda(Z)$. We say that a conjunction (disjunction) is *contracted* if no conjunct (disjunct, respectively) occurs twice in it. For $\varphi, \psi \in \mathsf{Pos}(Z)$, we say that φ *propositionally entails* ψ and write $\varphi \vdash_{PL} \psi$ if $\varphi \to \psi$ is a propositional tautology. Similarly, $\Phi \subseteq \mathsf{Pos}(Z)$ propositionally entails ψ ($\Phi \vdash_{PL} \psi$) if there exist $\varphi_1, \dots, \varphi_n \in \Phi$ such that $\varphi_1 \wedge \dots \wedge \varphi_n \vdash_{PL} \psi$. For $\varphi \in \mathsf{Pos}(Z)$, we denote the evaluation of φ in the Boolean algebra $\mathcal{P}(X)$ under a valuation $\tau : Z \to \mathcal{P}(X)$ by $[\![\varphi]\!]_{X,\tau}$, and write $X, \tau \models \varphi$ if $[\![\varphi]\!]_{X,\tau} = X$. For $\psi \in \mathsf{Pos}(\Lambda(Z))$, the interpretation $[\![\psi]\!]_{TX,\tau}$ of ψ in the Boolean algebra $\mathcal{P}(TX)$ under τ is the inductive extension of the assignment $[\![\heartsuit(z)]\!]_{TX,\tau} = [\![\heartsuit]\!]_X\tau(z)$. We write $TX, \tau \models \psi$ if $[\![\psi]\!]_{TX,\tau} = TX$.

We can now give the formal definition of the modal rule format, where due to monotonicity of the modal operators we can restrict to monotone rules following [4]. To understand the following, note that every rule of the form φ/χ, which says that if φ is provable then χ is provable, comes with a dual *tableau rule* $\overline{\chi}/\overline{\varphi}$ saying that if $\overline{\chi}$ is consistent then $\overline{\varphi}$ is consistent.

Definition 3. A *(monotone one-step) rule* $R = \varphi/\chi$ consists of a *premise* $\varphi \in \mathsf{Pos}(V)$ and a *conclusion* χ which is a disjunction over $\Lambda(V)$ (recall that V is the set of variables), where every variable appears at most once in φ and every variable in φ appears also in χ. The rule R is *one-step sound* if whenever $X, \tau \models \varphi$ for a valuation $\tau : V \to \mathcal{P}(X)$, then $TX, \tau \models \chi$. Given a set $\mathcal{R}$ of one-step rules, we say that a conjunction ψ over $\Lambda(V)$ is *one-step cut-free τ-consistent* for a set X and $\tau : V \to \mathcal{P}(X)$ if whenever

$\varphi/\chi \in \mathcal{R}$ and $\sigma : V \to V$ is a renaming such that $\chi\sigma$ is contracted and $\psi \vdash_{PL} \overline{\chi}\sigma$ (note that propositional entailment between conjunctions is just reverse containment), then $[\![\overline{\varphi}\sigma]\!]_{X,\tau} \neq \emptyset$. We say that $\mathcal{R}$ is *one-step cutfree complete* if $[\![\psi]\!]_{TX,\tau} \neq \emptyset$ whenever ψ is one-step cut-free τ-consistent. A set $\Psi \subseteq \Lambda(V)$ is *one-step cut-free τ-consistent* if for all $\psi_1, \dots, \psi_n \in \Psi$, $\psi_1 \wedge \dots \wedge \psi_n$ is one-step cut-free τ-consistent.

(In the terminology of [29], one-step cutfree complete rule sets correspond to one-step complete rule sets which are closed under contraction and resolution.) As the last parameter of the framework, we *fix from now on a one-step cutfree complete set $\mathcal{R}$ of one-step sound monotone one-step rules, and denote the arising logic by $\mathcal{L}_\sharp$*. Rules $\varphi/\psi \in \mathcal{R}$ are applied in substituted form, i.e for every substitution σ, we may conclude $\psi\sigma$ from $\varphi\sigma$. It is easy to see that the arising parametrized deduction system is sound. As usual, we write $\vdash \varphi$ if φ is provable, and $\varphi \vdash \psi$ if $\vdash \varphi \to \psi$. We say that φ is *consistent* if $\neg\varphi$ is not provable. It has been shown that one-step cutfree complete rule sets engender complete cut-free sequent systems for the purely modal logic, and suitable rule systems have been exhibited for all logics of Example 1 and many more [29,22]. E.g., a one-step cutfree complete set of monotone one-step rules for K is

$$\frac{\bigvee_{i=1}^{n} a_i \vee b}{\bigvee_{i=1}^{n} \Diamond a_i \vee \Box b} \; (n \geq 0).$$

As a more complex example, we recall the one-step cutfree complete rule schema for graded operators [29], reformulated to fit the monotone rule format:

$$\frac{\sum_{i=1}^{n} -r_i(\neg a_i) + \sum_{j=1}^{m} s_j b_j \geq 0}{\bigvee_{i=1}^{n} \Box_{k_i} a_i \vee \bigvee_{j=1}^{m} \Diamond_{l_j} b_j},$$

where $n + m \geq 1$ and $r_1, \dots, r_n, s_1, \dots, s_m > 0$, subject to the side condition $\sum_{i=1}^{n} r_i(k_i + 1) \geq 1 + \sum_{j=1}^{m} s_j l_j$. Here, the premise represents a linear inequality between the characteristic functions of the a_i and the b_j, i.e. count s_j when b_j holds and $-r_i$ when a_i does not hold; this is easily seen to be expressible by a positive propositional formula (cf. [29]).

4 Constructive Fixed Points

Our next aim is to prove that the Lindenbaum algebra of $\mathcal{L}_\sharp$ is *constructive*, i.e. its fixed points can be iteratively approximated in ω steps. In terms of consistency of formulas, this means that whenever a formula of the form $\sharp_\gamma\varphi \wedge \psi$ is consistent, then already $\gamma^i(\varphi)(\bot) \wedge \psi$ is consistent for some $i < \omega$; this fact plays a pivotal role in our tableau model construction. We begin by introducing the requisite algebraic tools.

We define a *Λ-modal algebra* A as a Boolean algebra extended with a monotone operation $\heartsuit^A : A \to A$ for each $\heartsuit \in \Lambda$. In such an algebra, every modal formula $\varphi(v_1, \dots, v_n)$ is naturally interpreted as an operation $\varphi^A : A^n \to A$. Now we say that A *validates* a rule $R = \varphi/\psi$ if $\psi^A(a_1, \dots, a_n) = \top$ whenever $\varphi^A(a_1, \dots, a_n) = \top$. A *$\sharp$-algebra* is a Λ-algebra A that is endowed with operations $\sharp_\gamma^A$ and $\flat_\gamma^A$ for each $\gamma \in \Gamma$ such that for each $a \in A$, $\sharp_\gamma^A(a)$ is the least fixed point of the map $\gamma^A(a, -) : A \to A$

and $\flat^A_{\overline{\gamma}}(a)$ is the greatest fixed point of $\overline{\gamma}^A(a,-)$ (in particular, these fixed points *exist* in a $\sharp$-algebra). An $\mathcal{L}_\sharp$*-algebra* is a $\sharp$-algebra A that validates every rule R of our fixed set $\mathcal{R}$ of one-step rules. In the tradition of algebraic logic, the class of these algebras provides an algebraic encoding of the proof system.

More specifically, we will be interested in the *Lindenbaum algebra* $A(\mathcal{L}_\sharp)$ of our logic. As usual, this algebra is defined as the quotient of the formula/term algebra (or absolutely free algebra) under the congruence relation $\equiv$ of provable equivalence ($\varphi \equiv \psi$ iff $\varphi \leftrightarrow \psi$ is derivable). Observe that in a natural way, every sentence φ is interpreted as the element $\varphi^{A(\mathcal{L}_\sharp)} = [\varphi]$ of this algebra; we will mostly write φ rather than $[\varphi]$. *The Kozen-Park axiomatization ensures that $A(\mathcal{L}_\sharp)$ actually is an $\mathcal{L}_\sharp$-algebra*, and then of course, the *initial* $\mathcal{L}_\sharp$-algebra. In these terms, our target property is phrased as follows.

Definition 4. We say that $\gamma \in \Gamma$ is *constructive* if

$$\sharp_\gamma \varphi = \bigvee_{i<\omega} \gamma(\varphi)^i(\bot)$$

in the Lindenbaum algebra $A(\mathcal{L}_\sharp)$, i.e. if $\sharp_\gamma\varphi \vdash \psi$ whenever $\gamma(\varphi)^i(\bot) \vdash \psi$ for all $i < \omega$. If all $\gamma \in \Gamma$ are constructive, then $A(\mathcal{L}_\sharp)$ *is constructive*.

We explicitly state the dual formulation of this property:

Lemma 5. *Let γ be constructive. If $\sharp_\gamma\varphi \wedge \psi$ is consistent, then $\gamma(\varphi)^i(\bot) \wedge \psi$ is consistent for some $i < \omega$.*

The central tool for proving constructivity, introduced in [26] and featuring prominently in [27], is the notion of a finitary $\mathcal{O}$-adjoint:

Definition 6. We say that γ is an $\mathcal{O}$*-adjoint* if for all $\varphi, \psi \in \mathcal{F}_\sharp$, there exists a finite set $G_{\gamma(\varphi)}(\psi)$ of formulas such that for all $\rho \in \mathcal{F}_\sharp$,

$$\gamma(\varphi, \rho) \vdash \psi \text{ iff } \rho \vdash \chi \text{ for some } \chi \in G_{\gamma(\varphi)}(\psi),$$

i.e. $\gamma(\varphi,\rho) \leq \psi$ in $A(\mathcal{L}_\sharp)$ iff $\rho \leq \chi$ for some $\chi \in G_{\gamma(\varphi)}(\psi)$. Moreover, γ is a *finitary $\mathcal{O}$-adjoint* if $G_{\gamma(\varphi)}$ can be chosen such that for every ψ, the closure of ψ under $G_{\gamma(\varphi)}$, i.e. the smallest set $\mathcal{A}$ with $\psi \in \mathcal{A}$ and $\chi \in \mathcal{A} \Rightarrow G_{\gamma(\varphi)}(\chi) \subseteq \mathcal{A}$, is finite.

Lemma 7. *[26] Every finitary $\mathcal{O}$-adjoint is constructive.*

The first step in the proof of $\mathcal{O}$-adjointness for a large class of operators is a generalization of the rigidity lemma of [26]:

Lemma 8 (Rigidity). *Let ψ be a disjunction over $\Lambda(A(\mathcal{L}_\sharp))$. Then ψ is provable iff there exists a one-step rule φ/χ and a substitution σ such that $\varphi\sigma$ is provable, $\chi\sigma$ is contracted, and $\chi\sigma \vdash_{PL} \psi$.*

The proof relies on the *one-point extension* of an algebra (so called because it mimics the addition of a new root point in a coalgebraic model on the algebraic side), in generalization of a similar construction in [27]:

Let A be a countable $\mathcal{L}_\sharp$-algebra, let $\mathcal{S}(A)$ be the set of ultrafilters of A, fix a surjective map $\sigma : V \to A$, and let a conjunction ρ over $\Lambda(V)$ be one-step θ-consistent for $\theta : V \to \mathcal{P}(\mathcal{S}(A))$ given by $\theta(v) = \{u \in \mathcal{S}(A) \mid \sigma(v) \in u\}$. We construct the one-point extension A^ρ, an $\mathcal{L}_\sharp$-algebra emulating the addition of a new point whose successor structure is described by ρ, as follows. To begin, we can find a maximally one-step θ-consistent set $\Phi \subseteq \Lambda(V)$ such that $\Phi \vdash_{PL} \rho$. As we emulate adding a single point, the carrier of A^ρ is $A \times 2$. We make A^ρ into a Λ-modal algebra by putting

$$\heartsuit^{A^\rho}(a, d) = (\heartsuit^A(a), \heartsuit^\rho(a)),$$

where $\heartsuit^\rho : A \to 2$ is defined by $\heartsuit^\rho(a) = \top$ iff $\heartsuit a \in \Phi\sigma$. (Thus, $\heartsuit^{A^\rho}(a, d)$ is independent of d, in agreement with the intuition that the interpretation of modal operators depends only on the successor structure of the current state, not on the state itself.) In particular, this implies that $\rho\sigma > \bot$ in A^ρ.

Lemma 9. *The algebra A^ρ is an $\mathcal{L}_\sharp$-algebra.*

In consequence of the fact that $A(\mathcal{L}_\sharp)$ is the *initial* $\mathcal{L}_\sharp$-algebra, we thus have

Lemma 10. *Let $\sigma : V \to A(\mathcal{L}_\sharp)$ be surjective. If a conjunction ρ over $\Lambda(\mathcal{F}_\sharp)$ is one-step θ-consistent for $\theta(v) = \{u \in \mathcal{S}(A(\mathcal{L}_\sharp)) \mid \sigma(v) \in u\}$, then ρ is consistent, i.e. $\rho > \bot$ in $A(\mathcal{L}_\sharp)$.*

From Lemma 10, one easily proves Lemma 8 using the fact that every consistent formula is contained in some ultrafilter of $A(\mathcal{L}_\sharp)$.

In a nutshell, rigidity enables us to prove $\mathcal{O}$-adjointness of all (monotone) modal operators, and even more generally all modal formulas where the argument variable x occurs at uniform depth (such as $\Box\Diamond x \wedge \Diamond\Box x$). Formally:

Definition 11. A formula φ with variables is *uniform of depth k* if every occurrence of the fixed argument variable x in φ is in the scope of exactly k modal operators (including the case that x does not occur in φ; recall moreover that variables never occur under fixed point operators). Moreover, φ is *uniform* if φ is uniform of depth k for some k; the minimal such k is the *depth of uniformity* of φ.

Finitaryness of $\mathcal{O}$-adjoints will use the standard Fischer-Ladner closure:

Definition 12. A set Σ of formulas is *Fischer-Ladner closed* if Σ is closed under subformulas and negation, and whenever $\star_\gamma\varphi \in \Sigma$, then $\gamma(\varphi, \star_\gamma\varphi) \in \Sigma$ for $\star \in \{\sharp, \flat\}$. We denote the Fischer-Ladner closure of a formula φ by $FL(\varphi)$.

Lemma 13. *[14] The set $FL(\varphi)$ is finite and of polynomial size in φ.*

The further development revolves largely around *admissible rules*, i.e. rules φ/ψ where φ and ψ are formulas with variables $v_1, \dots, v_n$ such that $A(\mathcal{L}_\sharp)$ validates φ/ψ, i.e. whenever $\vdash \varphi(\rho_1, \dots, \rho_n)$ for formulas $\rho_1, \dots, \rho_n$ then $\vdash \psi(\rho_1, \dots, \rho_n)$.

Lemma 14. *Let ψ be uniform, and put*

$$G = \{\varphi \in \mathsf{Pos}(FL(\psi)) \mid \varphi/\psi \textit{ admissible}, \varphi \textit{ uniform of depth } 0\}.$$

Then we have that for all ρ, $\psi(\rho)$ is provable iff $\varphi(\rho)$ is provable for some $\varphi \in G$.

Proof (sketch). Induction over the depth of uniformity, with trivial base case, using rigidity (Lemma 8) in the inductive step. □

Theorem 15 (Finitary $\mathcal{O}$-adjointness). *If the formula ψ with argument variable x is monotone and uniform in x, then the operation $\psi^{A(\mathcal{L}_\sharp)} : A(\mathcal{L}_\sharp) \to A(\mathcal{L}_\sharp)$ induced by ψ is a finitary $\mathcal{O}$-adjoint.*

Proof (sketch). For $\varphi \in \mathcal{F}_\sharp$, we have to construct a set $G_\psi(\varphi)$ of formulas such that for all $\rho \in \mathcal{F}_\sharp$, $\psi(\rho) \vdash \varphi$ iff $\rho \vdash \chi$ for some $\chi \in G_\psi(\varphi)$. Now $\psi' :\equiv \psi \to \varphi$ is uniform. Let $G \subseteq \mathsf{Pos}(FL(\psi'))$ be as in Lemma 14, applied to ψ', and let G_0 be a finite set of representatives of G modulo propositional equivalence. Then we can put

$$G_\psi(\varphi) = \{\chi(\top) \mid \chi \in G_0, \vdash \chi(\top) \vee \chi(\bot)\}.$$

□

Using uniform formulas as a base, we can now exploit some known closure properties of finitary $\mathcal{O}$-adjoints [26].

Definition 16. The set of *admissible* modal formulas is the closure of the set of monotone uniform modal formulas in x under disjunction, conjunction with modal formulas not containing x, and substitution for the argument variable, the latter in the sense that if γ and δ are admissible, then $\gamma(\delta)$ is admissible.

Corollary 17. *If $\gamma \in \Gamma$ is admissible, then γ is a finitary $\mathcal{O}$-adjoint, and hence constructive.*

From now on, *we require that every $\gamma \in \Gamma$ is admissible*, and hence $A(\mathcal{L}_\sharp)$ is constructive. All fixpoint operators mentioned in Example 1 are based on admissible formulas (in fact, on uniform ones).

5 The Tableau Construction

We proceed to describe a construction of *timed-out tableaux* for consistent formulas, which we shall then use as carrier sets for coalgebraic models. (Note that in coalgebraic logics, tableaux, being only relational structures, cannot directly serve as models.) Our time-outs are related to Kozen's μ-counters [14] but are integrated into the formulas appearing in the tableau (rather than maintained independently in the *construction* of the tableau), and in particular govern the way modal successor nodes are generated. The use of time-outs is justified by constructivity of fixed point operators as proved in the previous section. In the following, we fix a finite Fischer-Ladner closed set Σ.

Definition 18. The set of *timed-out formulas* φ, ψ is generated by the grammar

$$\varphi, \psi ::= \bot \mid \top \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \heartsuit\varphi \mid \sharp_\gamma(\rho)^\kappa \mid \flat_{\overline{\gamma}}(\rho) \qquad (\kappa \in \omega + 1, \rho \in \mathcal{L}_\sharp)$$

where $\gamma \in \Gamma, \heartsuit \in \Lambda$, subject to the restriction that φ is a timed-out formula only in case φ has at most one subformula of the form $\sharp_\gamma(\chi)^\kappa$ with $\kappa < \omega$ (which however may occur any number of times), and for this $\sharp_\gamma(\chi)^\kappa$, (i) $\sharp_\gamma(\chi)^\omega$ is not a subformula of φ; and (ii) whenever $\sharp_\delta(\rho)^\omega$ is a subformula of φ, then $\sharp_\delta(\rho)$ is a subformula of χ.

In this case, we define the *time-out* $\tau(\varphi)$ of φ to be κ, and $\tau(\varphi) = \omega$ otherwise (i.e. if φ does not contain any subformula of the form $\sharp_\gamma(\chi)^\kappa$ with $\kappa < \omega$). The time-out gives the number of steps left until satisfaction of the eventuality $\sharp_\gamma(\chi)$, with time-out ω signifying an unspecified number of steps (note that time-outs are never associated with $\flat$-formulas).

We define two translations s and t of timed-out formulas into $\mathcal{L}_\sharp$, given by commutation with Boolean and modal operators, $(\flat_{\overline{\gamma}}(\rho))^s = (\flat_{\overline{\gamma}}(\rho))^t = \flat_{\overline{\gamma}}(\rho)$, and

$$(\sharp_\gamma(\rho)^\omega)^s = \sharp_\gamma(\rho) \qquad (\sharp_\gamma(\rho)^i)^s = \gamma(\rho)^i(\bot) \quad (i < \omega) \qquad (\sharp_\gamma(\rho)^\kappa)^t = \sharp_\gamma(\rho).$$

Thus, s unfolds fixed points as prescribed by their time-outs, and t just removes time-outs. Both translations extend to sets of formulas. For timed-out formulas φ, ψ, we put $\varphi \preceq \psi$ iff $\varphi^t = \psi^t$ and $\tau(\varphi) \leq \tau(\psi)$. That is, $\varphi \preceq \psi$ iff φ is the same as ψ up to possible decrease of the time-out. Given a set Σ of formulas, a timed-out formula φ is a *timed-out Σ-formula* if $\varphi^t \in \Sigma$.

The point of the definition of timed-out formulas is that every standard formula φ has at most one candidate subformula at which one can insert a time-out, namely the greatest element under the subformula ordering among the subformulas of φ which are $\sharp$-formulas, if such a greatest element exists and is not under the scope of a $\flat$-operator. This enables the simple definition of $\preceq$, which trivially has the following property.

Lemma 19. *For every formula φ, the preimage of φ under the translation t is linearly ordered by $\preceq$.*

At the same time, timed-out formulas are stable under unfolding:

Lemma 20. *If $\sharp_\gamma\varphi^\kappa$ is a timed-out formula, then so is $\gamma(\varphi, \sharp_\gamma\varphi^\kappa)$.*

States of the tableau will be labelled by sets of formulas satisfying a timed-out version of the usual expandedness requirement.

Definition 21. A *timed-out Σ-atom* is a maximal set A of timed-out Σ-formulas such that (i) the translation t is injective on A, and (ii) A^s is consistent. Here, maximality is w.r.t. $\sqsubseteq$ where $A \sqsubseteq B$ iff for all $\varphi \in A$, there exists a (necessarily unique) $\varphi' \in B$ such that $\varphi' \preceq \varphi$; intuitively: B contains A up to possible decrease of time-outs. We write $\bar{A}$ for the closure of A under $\preceq$ (i.e. if $\varphi \in \bar{A}$ and $\varphi \preceq \varphi'$ then $\varphi' \in \bar{A}$).

The following lemma uses the fact that finite product orderings $(\omega+1)^k$ are well-quasi-orders, and in particular have only finite anti-chains [18].

Lemma 22. *The set of timed-out Σ-atoms is finite.*

Lemma 23 (Timed-out Lindenbaum lemma). *For every set A_0 of timed-out Σ-formulas such that A_0^s is consistent and t is injective on A_0, there exists a timed-out Σ-atom A such that $A_0 \sqsubseteq A$.*

The proof of the truth lemma crucially depends on a set of Hintikka-like properties:

Lemma 24. *If A is a timed-out Σ-atom, then*

1. *if* $\varphi \wedge \psi \in A$ *then* $\varphi \in \bar{A}$ *and* $\psi \in \bar{A}$*;*
2. *if* $\varphi \vee \psi \in A$ *then* $\varphi \in \bar{A}$ *or* $\psi \in \bar{A}$*;*
3. $\bot \notin \bar{A}$*;*
4. *if* $\sharp_\gamma \varphi^\kappa \in A$*, then* $\kappa < \omega$*;*
5. $\sharp_\gamma \varphi^\kappa \in A$ *iff* $\gamma(\varphi, \sharp_\gamma \varphi^{\kappa-1}) \in A$*;*
6. $\flat_\gamma \varphi \in A$ *iff* $\gamma(\varphi, \flat_\gamma \varphi) \in A$*.*

We proceed to define the actual tableaux, which relate timed-out atoms in a way that reflects application of dual rules $\overline{\chi}/\overline{\varphi}$ of modal rules $\varphi/\chi \in \mathcal{R}$, while fixed points are in a sense taken care of by the timed-out atoms themselves.

Definition 25. A *demand* of a Σ-atom A is a formula $\rho \equiv \bar{\varphi}\sigma$, where $\varphi/\chi \in \mathcal{R}$ is a rule with dual rule $\overline{\chi}/\overline{\varphi}$ and σ is a substitution such that $\chi\sigma$ is contracted and $A \vdash_{PL} \overline{\chi}\sigma$. A *timed-out* Σ*-tableau* $(\mathcal{T}, R, l)$ consists of a finite graph $(\mathcal{T}, R)$ and a labelling l of the nodes $n \in \mathcal{T}$ with timed-out Σ-atoms $l(n)$ such that for every demand ρ of $l(n)$, there exists nRm such that $l(m) \vdash_{PL} \rho$. The tableau $(\mathcal{T}, R, l)$ is a *timed-out* Σ*-tableau for* $\varphi \in \Sigma$ if $(\varphi')^t = \varphi$ for some $\varphi' \in l(n), n \in \mathcal{T}$. A coalgebra structure ξ on $\mathcal{T}$ is *coherent* if for every n and every $\heartsuit\varphi \in \Sigma$,

$$\xi(n) \in [\![\heartsuit]\!] n(\varphi) \text{ iff } \heartsuit\varphi \in l(n),$$

where $n(\varphi) = \{m \in \mathcal{T} \mid nRm, \varphi \in l(m)\}$.

The link between timed-out tableaux and coalgebraic models is provided by the following lemma, whose proof relies on one-step cutfree completeness of the rule set.

Lemma 26 (Model existence lemma). *For every timed-out* Σ*-tableau* $(\mathcal{T}, R, l)$*, there exists a coherent coalgebra structure on* $\mathcal{T}$*.*

Lemma 27 (Truth lemma). *If* $(\mathcal{T}, R, l)$ *is a timed-out* Σ*-tableau and* ξ *is a coherent coalgebra structure on* $\mathcal{T}$*, then* $n \in [\![\varphi]\!]_{(\mathcal{T},\xi)}$ *whenever* $\varphi \in l(n)$*.*

Proof (sketch). Induction over timed-out Σ-formulas φ using the lexicographic product of the subterm ordering on φ^t and $\preceq$ as the induction measure, and with the inductive hypothesis strengthened to apply also to $\varphi \in \overline{l(n)}$. Boolean cases are by Lemma 24; the step for modal operators is by coherence. The case for $\flat$-operators is by coinduction. For $\varphi = \sharp_\gamma(\psi)^\kappa$, we have $\kappa < \omega$ and $\gamma(\psi, \sharp_\gamma(\psi)^{\kappa-1}) \in \overline{l(n)}$ by Lemma 24. Then prove by a further induction over subformulas δ of γ that $n \models_{(\mathcal{T},\xi)} (\delta(\psi, \sharp_\gamma(\psi)^{\kappa-1}))^s$ whenever $\delta(\psi, \sharp_\gamma(\psi)^{\kappa-1}) \in \overline{l(n)}$. Here, the case for the parameter variable x is discharged by the inductive hypothesis applied to $\sharp_\gamma(\psi)^{\kappa-1}$. □

The previous two lemmas imply that every formula that has a timed-out tableau is satisfiable. The following lemma provides the link to consistency.

Lemma 28. *For any consistent* $\varphi \in \Sigma$ *there is a finite timed-out* Σ*-tableau.*

In summary, we have proved completeness of the Kozen-Park axiomatization:

Theorem 29 (Completeness). *If* Γ *is admissible and* $\mathcal{R}$ *is one-step cutfree complete, then the* $\mathcal{L}_\sharp$ *is complete over finite models.*

This result applies to all flat fixed point logics of Example 1, including all admissible flat fragments of AMC and the graded μ-calculus.

6 Complexity

Next we analyse the algorithmic aspects of satisfiability checking. This analysis is independent of the completeness result from Section 5 (except that completeness tells us that satisfiability checking is equivalent to consistency checking) but uses the same model construction. The complexity of the satisfiability problem as such is known: under additional conditions that we shall use below as well, it has been shown that satisfiability in the coalgebraic μ-calculus is in EXPTIME [4] (and therefore typically EXPTIME-complete, with hardness inherited from the standard μ-calculus). However, like known decision procedures for the standard μ-calculus, the algorithm in [4] uses automata-based methods and as such will exhibit exponential average-case behaviour, while a simple tableau method such as the one developed in Section 5 offers the possibility of feasible average-case behaviour using bottom-up construction of tableaux.

What is missing technically from the tableau construction of Section 5 with a view to complexity bounds is a bound on the time-outs. While we are confident that this can be proved directly using the $\mathcal{O}$-adjointness method (e.g. it is easy to show in this way that in Lemma 5, i can be exponentially bounded in $(\sharp_\gamma \varphi) \wedge \psi$), this is not actually necessary given that it has already been proved in [4] that the coalgebraic μ-calculus has the exponential model property. This implies immediately that time-outs can be exponentially bounded, so that tableaux are at most exponentially large. The key contribution of our tableaux construction here is to make this straightforward idea (which is similar in spirit to, e.g., Kozen's tableaux for the aconjunctive fragment of the μ-calculus [14]) work in a way that handles time-outs economically and consistently.

The size bound on tableaux alone does not yet imply an EXPTIME bound; however, we can obtain such a bound by using the coalgebraic generalization of the global caching method in exactly the same way as done in [13] for coalgebraic modal logic with global assumptions. To this end, we need to assume, as in [13,31], that our set $\mathcal{R}$ of one-step rules is EXPTIME-*tractable*, i.e. that there exists a coding of the rules such that, up to propositional equivalence, all demands of a conjunction over $\Lambda(\mathcal{F}_\sharp)$ can be generated by rules with codes of polynomially bounded size, and such that validity of codes, matching of rule codes for $\varphi/\chi \in \mathcal{R}$ to conjunctions ψ over $\Lambda(\mathcal{F}_\sharp)$ (in the sense of finding σ such that $\chi\sigma$ is contracted and $\psi \vdash_{PL} \overline{\chi}\sigma$), and membership of disjunctions in a CNF of a rule premise are all decidable in EXPTIME. Summing up,

> *if $\mathcal{R}$ is* EXPTIME-*tractable, then global caching decides existence of tableaux for $\mathcal{L}_\sharp$ in* EXPTIME.

Global caching will typically avoid full expansion of tableaux, and provides a handle to achieve feasible average-case performance using suitable heuristics.

7 Conclusions

We have raised the theory of flat modal fixed point logics [27] to the level of generality of coalgebraic logic. Specifically, we have given a Kozen-Park style axiomatization for fixed point operators, and we have shown this axiomatization to be sound and complete under the conditions that (i) the defining formulas of the fixed point operators satisfy a mild syntactic criterion, and (ii) the coalgebraic base logic is axiomatized by a one-step cutfree complete rule set. This result is a wide generalization with respect to the case of relational semantics, and covers, e.g., natural fixed point extensions of probabilistic modal logic and monotone modal logic. Most notably, we prove completeness of flat fragments of the graded μ-calculus [16], to our knowledge the first completeness result for any graded fixed point logic, and we generalize completeness of alternating-time temporal logic [12] to flat fragments of the alternating-time μ-calculus [1].

A core technical point in the proof was to show that essentially all monotone modal operators (including nested ones like $\Box\Box$, as long as the nesting depth is uniform) are finitary $\mathcal{O}$-adjoints in the sense of [26], and hence induce *constructive* fixed point operators that can be approximated in ω steps. This has enabled a model construction using tableaux with explicit time-outs for least fixed point formulas in the spirit of [14], which relies on a judicious definition of timed-out formula. As a byproduct of this construction, we obtain an optimal (i.e. EXPTIME) tableau calculus which paves the way for efficient implementations of coalgebraic flat fixed point logics, e.g. in the framework of the Coalgebraic Logic Satisfiability Solver CoLoSS [3].

Remaining open problems include the extension of the completeness result to larger fragments of the coalgebraic μ-calculus beyond the single variable fragment covered here, first and foremost the alternation-free fragment, and eventually the full coalgebraic μ-calculus. Similarly, there is the perspective to extend our tableau construction to at least the alternation-free fragment. A further direction for future research includes the development of generic coalgebraic model checking techniques.

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM 49, 672–713 (2002)
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook. Cambridge University Press, Cambridge (2003)
3. Calin, G., Myers, R., Pattinson, D., Schröder, L.: CoLoSS: The Coalgebraic Logic Satisfiability Solver (system description). In: Methods for Modalities (M4M-5, 2007). ENTCS, vol. 231, pp. 41–54. Elsevier, Amsterdam (2009)
4. Cîrstea, C., Kupke, C., Pattinson, D.: EXPTIME tableaux for the coalgebraic μ-calculus. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 179–193. Springer, Heidelberg (2009)
5. D'Agostino, G., Visser, A.: Finality regained: A coalgebraic study of Scott-sets and multisets. Arch. Math. Logic 41, 267–298 (2002)
6. Dam, M.: CTL* and ECTL* as fragments of the modal mu-calculus. Theoret. Comput. Sci. 126, 77–96 (1994)
7. Emerson, E.A., Clarke, E.M.: Using branching time temporal logic to synthesize synchronization skeletons. Sci. Comput. Program 2(3), 241–266 (1982)
8. Emerson, E.A., Halpern, J.Y.: Decision procedures and expressiveness in the temporal logic of branching time. J. Comput. System Sci. 30, 1–24 (1985)
9. Emerson, E.A., Jutla, C.S.: The complexity of tree automata and logics of programs. SIAM J. Comput. 29, 132–158 (1999)
10. Emerson, E.A., Lei, C.-L.: Efficient model checking in fragments of the propositional mu-calculus. In: Logic in Computer Science, LICS 1986, pp. 267–278. IEEE, Los Alamitos (1986)

11. Fine, K.: In so many possible worlds. Notre Dame J. Formal Logic 13, 516–520 (1972)
12. Goranko, V., van Drimmelen, G.: Complete axiomatization and decidability of alternating-time temporal logic. Theoret. Comput. Sci. 353, 93–117 (2006)
13. Goré, R., Kupke, C., Pattinson, D.: Optimal tableau algorithms for coalgebraic logics. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 114–128. Springer, Heidelberg (2010)
14. Kozen, D.: Results on the propositional μ-calculus. Theoret. Comput. Sci. 27, 333–354 (1983)
15. Kozen, D., Parikh, R.: An elementary proof of the completeness of PDL. Theoret. Comput. Sci. 14, 113–118 (1981)
16. Kupferman, O., Sattler, U., Vardi, M.Y.: The complexity of the graded μ-calculus. In: Voronkov, A. (ed.) CADE 2002. LNCS (LNAI), vol. 2392, pp. 423–437. Springer, Heidelberg (2002)
17. Larsen, K., Skou, A.: Bisimulation through probabilistic testing. Inf. Comput. 94, 1–28 (1991)
18. Laver, R.: Well-quasi-orders and sets of finite sequences. Math. Proc. Cambridge Philos. Soc. 79, 1–10 (1976)
19. Lewis, D.: Convention, A Philosophical Study. Harvard University Press, Cambridge (1969)
20. Parikh, R.: The logic of games and its applications. Annals of Discrete Mathematics 24, 111–140 (1985)
21. Pattinson, D.: Expressive logics for coalgebras via terminal sequence induction. Notre Dame J. Formal Logic 45, 19–33 (2004)
22. Pattinson, D., Schröder, L.: Cut elimination in coalgebraic logics. Inf. Comput. (to appear)
23. Pauly, M.: A modal logic for coalitional power in games. J. Log. Comput. 12, 149–166 (2002)
24. Peleg, D.: Concurrent dynamic logic. J. ACM 34, 450–479 (1987)
25. Pratt, V.R.: Semantical considerations on Floyd-Hoare logic. In: Foundations of Computer Science, FOCS 1976, pp. 109–121. IEEE, Los Alamitos (1976)
26. Santocanale, L.: Completions of μ-algebras. Ann. Pure Appl. Logic 154, 27–50 (2008)
27. Santocanale, L., Venema, Y.: Completeness for flat modal fixpoint logics. Annals Pure Appl. Log. (preprint under) (to appear), http://arxiv.org/abs/0812.2390
28. Schröder, L.: A finite model construction for coalgebraic modal logic. J. Log. Algebr. Prog. 73, 97–110 (2007)
29. Schröder, L., Pattinson, D.: PSPACE bounds for rank-1 modal logics. ACM Trans. Comput. Log. 10, 13:1–13:33 (2009)
30. Schröder, L., Pattinson, D.: Strong completeness of coalgebraic modal logics. In: Theoretical Aspects of Computer Science, STACS 2009, pp. 673–684. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl (2009)
31. Schröder, L., Pattinson, D., Kupke, C.: Nominals for everyone. In: International Joint Conferences on Artificial Intelligence, IJCAI 2009, pp. 917–922. AAAI Press, Menlo Park (2009)
32. Segerberg, K.: A completeness theorem in the modal logic of programs. In: Universal Algebra and Applications. Banach Centre Publications, vol. 9, pp. 31–46. PWN (1982)
33. Walukiewicz, I.: Completeness of Kozen's axiomatisation of the propositional μ-calculus. Inf. Comput. 157, 142–182 (2000)

Conditional Automata: A Tool for Safe Removal of Negligible Events

Roberto Segala and Andrea Turrini

Dipartimento di Informatica
Università di Verona - Italy

Abstract. Polynomially accurate simulations [19] are relations for Probabilistic Automata that require transitions to be matched up to negligible sets provided that computation lengths are polynomially bounded. They are proposed for verification of cryptographic protocols. In this paper we introduce a general construction on probabilistic automata, called Conditional Automata, that allows us to remove safely events that occur with negligible probability. The construction is justified in terms of polynomially accurate simulations. This, combined with the hierarchical and compositional verification style that underlies simulation relations, permits one to abstract one cryptographic component at a time in a complex system. We illustrate our construction through a simple example based on nonce generation, where we remove the event of repeated nonces.

1 Introduction

Hierarchical and compositional verification is one of the most successful byproducts of concurrency theory: compositional verification allows one to decompose complex systems into several sub-components that can be analyzed separately, while hierarchical verification permits one to analyze the interaction between several components on appropriate abstractions that do not include any irrelevant detail, thus reducing the overall complexity of a verification task.

Recently there has been increasing interest in hierarchical and/or compositional verification methods for security protocols [4,5,14,16,1,7]. Indeed, a generic security protocol runs in an arbitrarily large network of agents and involves the interaction of several instances of functionalities like encryption schemes, signature schemes, nonce generators, etc. The resulting systems are rather complex to analyze, and therefore the ability to analyze components separately turns out to be a significant relief in the verification task and may allow us to reuse old proofs in other contexts. This approach, although of independent interest, turns out to be useful in automatic verification as well since it allows us to limit the well known state explosion problem.

In [19] we have used Probabilistic Automata [17] and proposed *polynomially accurate simulations* (PASs) as a tool for verifying hierarchically cryptographic protocols. The main idea is to adapt the simulation method of [11], already used in the context of distributed systems, to the area of security. Specifically, rather than requiring transitions of a concrete implementation to be matched exactly

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 539–553, 2010.

by the abstract system, we allow for errors of negligible probability. This allows us to combine abstraction steps that rely on cryptographic assumptions, justified in terms of PASs, with abstraction steps that do not rely on any cryptographic assumption and that can be justified in terms of classical concurrency theory.

Polynomially accurate simulations are used successfully in [21] for the analysis of protocols, including a complete formal proof of the Dolev-Yao soundness result of [9]. In the case studies of [21] we start with a concrete description of the protocol under analysis, we abstract away the undesirable behaviors that occur with negligible probability, and then we proceed with other abstraction steps to prove that the resulting system always satisfies the protocol specification.

In this paper we focus on a general technique, used in [21], that allows us to remove negligible events from a concrete system while preserving all properties that hold with overwhelming probability in any context. For instance, if we have a complex system that obtains nonces from a nonce generator that draws random numbers, then we know that the probability of obtaining repeated nonces is negligible. It is then reasonable to analyze the complex system assuming that repeated nonces never occur. What is needed, however, is a theorem that supports our reasonable proposal. Let $\mathcal{A}$ be a probabilistic automaton describing the nonce generator that draws random numbers. Then repeated nonces may occur with negligible probability. Let G be the set of *good* states of $\mathcal{A}$ where no nonce is repeated. We introduce the concept of *conditional automaton* $\mathcal{A}|G$, obtained from $\mathcal{A}$ by replacing the target measure μ of each transition by the measure $\mu|G$, that is, the measure μ conditional on reaching a state from G. Our main theorem states that the identity relation is a PAS from $\mathcal{A}$ to $\mathcal{A}|G$ as well as from $\mathcal{A}|G$ to $\mathcal{A}$. The theorem holds for any set of states G whose complement is negligible. As a consequence, we can analyze systems hierarchically and remove safely any negligible event from any component by replacing it with its conditional counterpart.

Of course, although the final statement of he theorem turns out to be simple, we need to define precisely what we mean by negligible event in a probabilistic automaton, how we identify the set G of good states, and how PASs allow us to relate computations of a concrete system to computations of its abstraction. This is indeed the most complex part of the whole treatment. Informally, the notion of negligible event is defined by parameterizing probabilistic automata by a security parameter k and requiring bad states to have a negligible probability whenever computations are of polynomial length, G is identified by recording the past history in the states of an automaton, while correspondence between computations is stated by an *execution correspondence theorem* that turns out to be a generalization of the mapping lemma of [13].

One problem of the PASs of [19] is that, although they admit hierarchical verification, they are not compositional, that is, their existence is not preserved by parallel composition of probabilistic automata. On the other hand, a typical security protocol does not rely only on nonce generators, but rather it relies on encryption and/or signature schemes as well. Since also for encryption and signature schemes there are negligible events that we would like to remove (e.g., generation of repeated keys or generation of repeated ciphertexts in an IND-CCA schema), and since it is reasonable to apply conditional automata on each single primitive individually, compositionality of PASs becomes essential. For this reason, in the first part of the paper we introduce a new notion of PAS (state PAS or sPAS) that is stronger than the one of [19] and that is compositional as well. Roughly speaking, the PASs of [19] require matching of hyper-transitions up to negligible errors, while our new simulations require that the set of pairs of concrete and abstract states where the abstract states can match the transitions of the concrete state up to a negligible error has an overwhelming probability.

Related Work. The simulation method is used already in the security literature. For example in [4] bisimulation relations are used to prove correctness of implementations according to the notion of reactive simulatability [16]. Although the definition of bisimulation is not worked out in full details, the idea is clear: transitions should be matched up to some “error sets”, where an error set is a set of parts of transitions (e.g., messages, states) that have no corresponding piece in the abstract system; then, a separate argument shows that the global probability of the error sets is negligible. In our approach we impose conditions on the probabilities of the error sets directly in the step condition with the aim of bounding the global probability of the error sets to be negligible. Simulation relations are used also in [7] in the context of the Universally Composable framework [5]. In this case simulation relations are exact and the computational arguments are carried out with respect to a notion of approximated probabilistic language inclusion [6] based on the trace distribution semantics of [17]. Also in [14] there is a use of exact probabilistic bisimulations in the context of a probabilistic polynomial time process calculus. In this case the computational aspects are captured directly in the definition of the calculus. Another proposal of approximated probabilistic simulation relations appears in [15]. In this case a distance between probability of measures is defined based on the ability to produce similar trace distributions. Then an ε simulation matches steps from ε-distant measures by preserving ε-distance. Our definition is based on a different distance that may grow by a negligible amount at each step.

Though we are not aware of any formal treatment of the idea of conditional automaton in the literature, informal arguments along the lines of our results are common in the security community. In this context our result provides additional foundations and rigor to an idea that is common belief within the area of security. In [20] there is an idea of representing a correctness proof as a sequence of related games, where games are representations of attacks against protocols that are described at different levels of abstraction, and where two games are related if the difference of the probabilities of successful attacks is negligible. A general result of [20] is the *Difference Lemma* that implicitly gives us a way to transform a game G to a polynomially-equivalent one where some negligible failure event does not occur any more. Game transformations can be seen as an alternative hierarchical way of analyzing protocols. They are missing compositionality, though.

Of course almost all proofs of Dolev-Yao soundness (see, e.g., [13,3] and related literature) rely on some form of abstraction from negligible events. The aim of this paper is to provide a tool based on probabilistic automata and simulation relations that allows us to safely remove negligible events from generic automata in order to obtain basic building blocks that can be used to recast Dolev-Yao soundness proofs following a modular approach (see, e.g., our case study in [21]). For instance, conditional automata can be used to abstract away events like signature forgery and key guessing. The main advantage of conditional automata proposed in this paper is that our construction is general and the removal of negligible events is justified in terms of approximate simulations that support hierarchical and compositional verification. This allows us to split events depending on properties of cryptographic primitives from events arising from the interaction of adversary and participants of the protocol, to study an event at a time independently from the others, and to exploit proof techniques and results of concurrency theory in other fields like verification of cryptographic protocols.

The rest of the paper is structured as follows: Section 2 recalls Probabilistic Automata and gives some basic notions on nonce generation; Section 3 introduces state polynomially accurate simulations as well as the execution correspondence theorem; Section 4 defines conditional automata and states the conditional automaton theorem; Section 5 gives some concluding remarks.

The proof of the results proposed in this paper can be found in [21].

2 Preliminaries

2.1 Probabilistic Automata

Given a set X, denote by $\mathit{Disc}(X)$ the set of discrete probability measures over X, and by $\mathit{SubDisc}(X)$ the set of discrete sub-probability measures over X. We call a discrete probability measure a *Dirac* measure if it assigns measure 1 to the singleton set $\{x\}$ (denote this measure by δ_x). We also call Dirac a sub-probability measure that assigns measure 0 to all objects. In the sequel discrete sub-probability measures are used to describe progress. If the measure of X is not 1, then it means that with some non-zero probability the system does not progress. Given $\rho \in \mathit{SubDisc}(X)$, we denote by $\mathit{Supp}(\rho)$ the *support* set $\{x \in X \mid \rho(x) > 0\}$ of ρ.

Given a set X, a set $G \subseteq X$, and a measure $\rho \in \mathit{Disc}(X)$ such that $\rho(G) > 0$, we call the *G-conditional measure of* ρ, denoted by $\rho|G$, the probability measure $\rho' \in \mathit{Disc}(X)$ such that for each $x \in X$,

$$\rho'(x) = \begin{cases} \rho(x)/\rho(G) & \text{if } x \in G, \\ 0 & \text{otherwise.} \end{cases}$$

A *Probabilistic Automaton* (PA) is a tuple $(S, \bar{s}, A, D)$ where S is a set of *states*, $\bar{s} \in S$ is the *start state*, A is a set of *actions*, and $D \subseteq S \times A \times \mathit{Disc}(S)$ is a *transition relation*. The set of actions A is further partitioned into three sets I, O, H of input, output and internal (hidden) actions, respectively. We call the set $E = I \cup O$ the set of external actions.

Throughout the paper we let $\mathcal{A}$, $\mathcal{B}$ range over probabilistic automata, q, r, s range over states, a, b, c range over actions, and μ range over discrete measures over states. We also denote the generic elements of a probabilistic automaton $\mathcal{A}$ by S, $\bar{s}$, A, D, and we propagate primes and indices when necessary. Thus, for example, the probabilistic automaton $\mathcal{A}'_i$ has transition relation D'_i.

An element of a transition relation D is called a *transition* or a *step*. A transition $tr = (s, a, \mu)$, also denoted by $s \xrightarrow{a} \mu$, is said to *leave* from state s, to be *labeled* by a, and to *lead* to μ, denoted by μ_{tr}. We also say that state s *enables* action a, that action a is *enabled* from s, and that (s, a, μ) is enabled from s. We denote by $D(s)$ the set of all transitions enabled by s, by $D(a)$ the set of all transitions labeled by a, and by $D(sa)$ their intersection (that is, $D(sa) = D(s) \cap D(a)$).

An *execution fragment* of a PA $\mathcal{A}$ is a sequence of alternating states and actions, $\alpha = s_0 a_1 s_1 \ldots$, starting with a state and, if the sequence is finite, ending with a state, such that, for each non final index i, there exists a transition $(s_i, a_{i+1}, \mu_{i+1})$ in D with $\mu_{i+1}(s_{i+1}) > 0$. We say that an execution fragment is *finite* if it is a finite sequence, and we denote the last state of a finite execution fragment α by $\mathit{lstate}(\alpha)$. We define the length $|\alpha|$ of an execution fragment α to be the number of occurrences of actions in α. An *execution* of a PA $\mathcal{A}$ is an execution fragment of $\mathcal{A}$ whose first state is $\bar{s}$. We denote by $\mathit{Frags}^*(\mathcal{A})$ the set of finite execution fragments of $\mathcal{A}$, by $\mathit{Frags}(\mathcal{A})$ the set of finite or infinite execution fragments, and by $\mathit{Execs}^*(\mathcal{A})$, $\mathit{Execs}(\mathcal{A})$ the corresponding sets of executions.

A scheduler for a PA $\mathcal{A}$ is a function $\sigma\colon \mathit{Frags}^*(\mathcal{A}) \to \mathit{SubDisc}(D)$ such that, for each finite execution fragment α and each transition $tr = (s, a, \mu)$ with $\sigma(\alpha)(tr) > 0$, $s = \mathit{lstate}(\alpha)$. A scheduler σ and a state s induce a probability measure $\nu_{\sigma,s}$ over execution fragments as follows. The basic measurable events are the cones of finite execution fragments, where the cone of a finite execution fragment α, denoted by C_α, is the set $\{\alpha' \in \mathit{Frags}(\mathcal{A}) \mid \alpha \leqslant \alpha'\}$, where $\leqslant$ is the standard prefix preorder on sequences. The probability $\nu_{\sigma,s}$ of a cone C_α is defined recursively as follows:

$$\nu_{\sigma,s}(C_\alpha) = \begin{cases} 0 & \text{if } \alpha = q \text{ for some state } q \neq s, \\ 1 & \text{if } \alpha = s, \\ \nu_{\sigma,s}(C_{\alpha'}) \sum_{tr \in D(a)} \sigma(\alpha')(tr) \mu_{tr}(q) & \text{if } \alpha = \alpha' a q. \end{cases}$$

Standard measure theoretical arguments ensure that $\nu_{\sigma,s}$ extends uniquely to the σ-field generated by cones. If $s = \bar{s}$, we call the measure $\nu_{\sigma,\bar{s}}$ a *probabilistic execution* ν_σ of $\mathcal{A}$ and we say that it is generated by σ from $\bar{s}$.

Remark 1. A typical cryptographic protocol described in the literature does not have nondeterminism, and hence there is a unique maximal probabilistic execution, where by maximal we mean that we stop only when no more transitions are enabled. However, as soon as we abstract from some probabilistic choices nondeterminism may appear. Imagine, for example, to abstract at the Dolev-Yao level just some of the functionalities of a protocol: in this case, we have that choices of abstracted functionalities are performed nondeterministically while the choices of other functionalities are still probabilistic. Besides, there is already evidence in the recent literature [8] that sometimes it is not convenient to hide nondeterminism under the carpet.

We now turn to the notion of simulation for probabilistic automata [17], defining first what it means to lift a relation on states to a relation on measures. Let $\mathcal{R}$ be a relation from a set X to a set Y. The lifting of $\mathcal{R}$, denoted by $\mathcal{L}(\mathcal{R})$, is a relation from $\mathit{Disc}(X)$ to $\mathit{Disc}(Y)$ such that $\rho_X\ \mathcal{L}(\mathcal{R})\ \rho_Y$ if and only if there exists a *weighting function* $w\colon X \times Y \to [0, 1]$ such that (1) $w(x, y) > 0$ implies $x\ \mathcal{R}\ y$, (2) $\sum_{x \in X} w(x, y) = \rho_Y(y)$, and (3) $\sum_{y \in Y} w(x, y) = \rho_X(x)$. An alternative definition of lifting given in a more probabilistic style states that $\rho_X\ \mathcal{L}(\mathcal{R})\ \rho_Y$ if and only if there exists a joint measure w with marginal measures ρ_X and ρ_Y such that the support of w (i.e., pairs (x, y) such that $w(x, y) > 0$) is included in $\mathcal{R}$. If we view $\mathcal{R}$ as a pseudo-metric with values in $\{0, \infty\}$, then $\mathcal{L}(\mathcal{R})$ is the Kantorovich metric on probability measures [10].

A *simulation* from a PA $\mathcal{A}_1$ to a PA $\mathcal{A}_2$ is a relation $\mathcal{R}$ from S_1 to S_2 such that $\bar{s}_1\ \mathcal{R}\ \bar{s}_2$ and for each pair $(s_1, s_2) \in \mathcal{R}$, if $(s_1, a, \mu_1) \in D_1$, then there exists a transition $s_2 \xrightarrow{a} \mu_2$ such that $\mu_1\ \mathcal{L}(\mathcal{R})\ \mu_2$. We say that $\mathcal{A}_1$ is simulated by $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \preceq \mathcal{A}_2$, if there exists a simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$. Relation $\preceq$ is transitive and preserved by the parallel composition $\mathcal{A}||\mathcal{B}$ of PAs $\mathcal{A}$ and $\mathcal{B}$. This is the key feature that enables hierarchical and modular verification. We do not define formally composition here and we refer the interested reader to [18].

2.2 Nonces and Negligibility

A *nonce* of length k is an element of $\{0, 1\}^k$ that is used at most once. An ideal way to satisfy uniqueness of nonces is to use a repository that keeps track of the nonces distributed in the past and that responds to all requests by returning a new value each time. The practical way to satisfy the uniqueness of nonces is to choose them randomly from $\{0, 1\}^k$. In this way, if we choose randomly two nonces of length k, the probability that they are the same is at most 2^{-k}. This property can be extended to any polynomial number of chosen nonces, as follows: denoted by *Poly* the set of positive polynomials on $\mathbb{N}$,

Property 1. For each $c \in \mathbb{N}$ and $p \in \mathit{Poly}$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$, given $n_1, \dots, n_{p(k)} \in \{0, 1\}^k$, if n is chosen randomly and uniformly from $\{0, 1\}^k$, then $\mathbf{Pr}(n \in \{n_1, \dots, n_{p(k)}\}) < k^{-c}$.

The phrase "for each $c \in \mathbb{N}$ and $p \in \mathit{Poly}$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$" is commonly used in the literature on complexity theory when bounding the probability of an attack in the computational model. Essentially, this phrase means that no matter how we bound polynomially the number of actions the attacker can perform ($\forall p \in \mathit{Poly}$) and how we limit the polynomial probability of the attack ($\forall c \in \mathbb{N}$), we are able to find a minimum value of the security parameter ($\bar{k} \in \mathbb{N}$) such that for each larger value k the probability of the attack is less than the imposed limit k^{-c}, that is, the probability of the attack is less than each polynomial in k.

3 State Polynomially Accurate Simulation

3.1 State Polynomially Accurate Simulation

We start by enriching the notion of lifting of a relation to account for errors.

Definition 1. *Let $\mathcal{R}$ be a relation from X to Y and let $\varepsilon \in \mathbb{R}^{\geqslant 0}$. The ε-lifting of $\mathcal{R}$, denoted by $\mathcal{L}(\mathcal{R}, \varepsilon)$, is a relation from Disc(X) to Disc(Y) defined as follows: for each pair ρ_X, ρ_Y of measures in Disc(X) and Disc(Y), respectively, $\rho_X \; \mathcal{L}(\mathcal{R}, \varepsilon) \; \rho_Y$ if and only if either $\varepsilon \geqslant 1$ or there exists an* ε-weighting function $w_\varepsilon \colon X \times Y \to [0, 1]$ *such that*

1. $w_\varepsilon(x, y) > 0 \implies x \, \mathcal{R} \, y$,
2. $\sum_{y \in Y} w_\varepsilon(x, y) \leqslant \rho_X(x)$,
3. $\sum_{x \in X} w_\varepsilon(x, y) \leqslant \rho_Y(y)$,
4. $\sum_{x \in X, y \in Y} w_\varepsilon(x, y) \geqslant 1 - \varepsilon$.

Alternatively we can say that there exists a joint sub-probability measure w with marginal measures dominated by ρ_X and ρ_Y, total mass at least $1 - \varepsilon$, and such that the support of w is included in $\mathcal{R}$. Also, if we view $\mathcal{R}$ as a pseudo-metric with values in $\{0, 1\}$, then the supremum $1 - \varepsilon$ such that $\rho_X \; \mathcal{L}(\mathcal{R}, \varepsilon) \; \rho_Y$ holds is the Kantorovich distance between ρ_X and ρ_Y. Thus, testing $\rho_1 \; \mathcal{L}(\mathcal{R}, \varepsilon) \; \rho_2$ is equivalent to solving a maximum flow problem with the additional requirement that the flow is at least $1 - \varepsilon$. The original and equivalent definition given in [19] is also interesting: $\rho_X \; \mathcal{L}(\mathcal{R}, \varepsilon) \; \rho_Y$ iff either $\varepsilon \geqslant 1$ or there exist $\rho'_X, \rho''_X, \rho'_Y, \rho''_Y$ such that $\rho_X = (1 - \varepsilon)\rho'_X + \varepsilon\rho''_X$, $\rho_Y = (1 - \varepsilon)\rho'_Y + \varepsilon\rho''_Y$, and $\rho'_X \; \mathcal{L}(\mathcal{R}) \; \rho'_Y$.

The second ingredient for sPASs is the ε-step condition. Informally, two states s_1, s_2 satisfy the ε-step condition if each transition from s_1 can be matched by a transition from s_2 up to an ε-bounded error.

Definition 2. *Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two automata and $\mathcal{R}$ be a relation from S_1 to S_2. Given $s_1 \; \mathcal{R} \; s_2$, we say that s_1 and s_2 satisfies the* ε-step condition*, denoted by $s_1 \; St(\mathcal{R}, \varepsilon) \; s_2$, if for each $(s_1, a, \mu_1) \in D_1$, there exists a transition $s_2 \xrightarrow{a} \mu_2$ such that $\mu_1 \; \mathcal{L}(\mathcal{R}, \varepsilon) \; \mu_2$.*

For the third ingredient, consider a relation $\mathcal{R}$ and two measures ρ_1, ρ_2 that are $\mathcal{L}(\mathcal{R}, \gamma)$-related. We know that there is at least one γ-weighting function that supports $\rho_1 \; \mathcal{L}(\mathcal{R}, \gamma) \; \rho_2$. Indeed, there may be several such weighting functions. Among them we would like to select one that reduces as much as possible the mass of those pairs that are not $St(\mathcal{R}, \varepsilon)$-related. We first define formally the condition above by requiring the mass of the "bad" pairs to be bounded by η.

Definition 3. *Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two PAs and let $\mathcal{R}$ be a relation from S_1 to S_2. We say that $\mathcal{R}$ is an* ε-η-approximate simulation *if*

- $\bar{s}_1 \; \mathcal{R} \; \bar{s}_2$;
- *for each γ, μ_1, μ_2 such that $\mu_1 \; \mathcal{L}(\mathcal{R}, \gamma) \; \mu_2$ there exists a γ-weighting function w_γ for $\mu_1 \; \mathcal{L}(\mathcal{R}, \gamma) \; \mu_2$ such that $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg \; St(\mathcal{R}, \varepsilon) \; s_2\} < \eta$.*

Remark 2. The step condition of the ε-η-approximate simulation is not immediate. Indeed, the reader may wonder

- *why not replacing $\mathcal{L}(\mathcal{R})$ with $\mathcal{L}(\mathcal{R}, \varepsilon)$ in the step condition of an ordinary simulation?* Our objective, illustrate in Figure 1, is to take a probabilistic execution ν of a simulated automaton and produce a probabilistic execution of the simulating one that corresponds up to some error that grows with the length n of ν, but not too much. If we use a $\mathcal{L}(\mathcal{R}, \varepsilon)$ relation, then the error is bounded above by $1 - (1 - \varepsilon)^n$, that is, the error gets exponentially close to 1, which is too much;
- *what is the purpose of γ?* In order to avoid the error being exponentially close to 1, we propose a way to say that the error grows at most by fixed amount at each step. In our definition γ represents the error before simulating the current step, while η represents a bound on the extra error introduced by the step;
- *why ε-η?* A step may be simulated up to some small error either because there are sufficiently many pairs of states that simulate each other perfectly, or because all pairs of states simulate each other up to some small error, or because of a combination of the two. The parameter ε measures the second kind of error, while η measures the first kind of error. The combination of the two measures the third kind of error. Thus, for instance, if we set ε to be 0 we consider only the first kind of error, and if we set η to be 0 we consider only the second kind of error.

Observe that in the sum above the fact that $w_\gamma(s_1, s_2) > 0$ implies implicitly that $s_1 \mathcal{R} s_2$ as well as $s_1 \in \mathit{Supp}(\mu_1)$ and $s_2 \in \mathit{Supp}(\mu_2)$.

The final step to define sPASs is to enrich ε-η-approximate simulations with computational elements. Thus, we use families of PAs and of relations parameterized by a security parameter k, we consider only pairs of measures that are reached within polynomial time, and we require ε and η to be smaller than any polynomial. We use the notion of probability measure on states reachable within n steps. The formal definition, which requires some basic measure theory, states that μ is reachable within n steps if there exists a probabilistic execution, supported on executions of length at most n, whose image measure of the *lstate* function is μ.

Definition 4. *A measure μ on states is said to be* reachable within n steps *if there exists a probabilistic execution ν, supported on finite executions of length at most n, such that $\mu = \mathit{lstate}(\nu)$.*

Definition 5. *Let $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ and $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$ be two families of probabilistic automata; let $\mathcal{R}= \{\mathcal{R}_k\}_{k\in\mathbb{N}}$ be a family of relations such that each $\mathcal{R}_k$ is a relation from S_k^1 to S_k^2; let Poly be the set of positive polynomials over $\mathbb{N}$. We say that $\mathcal{R}$ is a* state polynomially accurate simulation *from $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ to $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$ if*

1. *for each* $k \in \mathbb{N}$, *it holds that* $\bar{s}^1_k \; \mathcal{R}_k \; \bar{s}^2_k$*;*
2. *for each* $c \in \mathbb{N}$ *and* $p \in \mathit{Poly}$, *there exists* $\bar{k} \in \mathbb{N}$ *such that for each* $k > \bar{k}$, *for all probability measures* μ_1 *and* μ_2 *and for each* $\gamma \geqslant 0$,
   - *if* μ_1 *is reached within* $p(k)$ *steps in* $\mathcal{A}^1_k$ *and* $\mu_1 \; \mathcal{L}(\mathcal{R}_k, \gamma) \; \mu_2$,
   - *then there exists a* γ*-weighting function* w_γ *for* $\mu_1 \; \mathcal{L}(\mathcal{R}_k, \gamma) \; \mu_2$ *such that* $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg \; \mathit{St}(\mathcal{R}_k, k^{-c}) \; s_2\} < k^{-c}$.

We write $\{\mathcal{A}^1_k\}_{k \in \mathbb{N}} \lesssim_s \{\mathcal{A}^2_k\}_{k \in \mathbb{N}}$ *if there exists a state polynomially accurate simulation* $\mathcal{R}$ *from* $\{\mathcal{A}^1_k\}_{k \in \mathbb{N}}$ *to* $\{\mathcal{A}^2_k\}_{k \in \mathbb{N}}$.

Remark 3. In the definition of the step condition, we use the same bound k^{-c} for two different quantities: the first one is the error ε of the ε-step condition, while the second one is the overall weight η of pair of states that do not satisfy the ε-step condition. The choice of using the same value is not restrictive: if we use two different bounds, say $k^{-c'}$ and $k^{-c''}$, then we can take $c = \min(c', c'')$ and the step condition still holds. In fact, it is easy to observe that $s_1 \neg \; \mathit{St}(\mathcal{R}_k, k^{-c}) \; s_2$ implies $s_1 \neg \; \mathit{St}(\mathcal{R}_k, k^{-c'}) \; s_2$ and thus $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg \; \mathit{St}(\mathcal{R}_k, k^{-c'}) \; s_2\} < k^{-c''}$ implies $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg \; \mathit{St}(\mathcal{R}_k, k^{-c}) \; s_2\} < k^{-c}$.

Notational convention. To simplify the notation, when it is clear from the context we denote the family of automata $\{\mathcal{A}_k\}_{k \in \mathbb{N}}$ by $\mathcal{A}$. We still use $\mathcal{A}_k$ to denote the automaton of the family $\{\mathcal{A}_k\}_{k \in \mathbb{N}}$ instantiated with the specific security parameter k. Similarly, we denote the family of relations $\{\mathcal{R}_k\}_{k \in \mathbb{N}}$ by $\mathcal{R}$.

Remark 4. We do not compare explicitly sPAS with the polynomially accurate simulations of [19] here due to lack of space. For the reader familiar with [19] we observe that assuming Definition 3 it is possible to prove that for any measures $\mu_1 \; \mathcal{L}(\mathcal{R}, \gamma) \; \mu_2$, any hyper-transition $\mu_1 \longrightarrow \rho_1$ can be matched by a hyper-transition $\mu_2 \longrightarrow \rho_2$ such that $\rho_1 \; \mathcal{L}(\mathcal{R}, \gamma + \varepsilon + \eta) \; \rho_2$. This is the basic step to show that sPASs are stronger than the PASs of [19].

Remark 5. It is immediate to observe that an ordinary simulation relation $\mathcal{R}$ is a special case of a sPAS since $s_1 \; \mathcal{R} \; s_2$ implies $s_1 \; \mathit{St}(\mathcal{R}, k^{-c}) \; s_2$. Thus, in a complex system it is safe to use ordinary simulations and basic concurrency theory on all those components that do not rely on any computational assumption.

Remark 6. Here we are working with relations that in concurrency theory literature are known as *strong*. However, the reader familiar with weak relations may note that extending our result is not difficult: just modify the condition of the step condition imposing that $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg \; \mathit{St}^w(\mathcal{R}_k, k^{-c}) \; s_2\} < k^{-c}$ where $s_1 \; \mathit{St}^w(\mathcal{R}_k, k^{-c}) \; s_2$ if for each transition (s_1, a, μ_1), there exists a weak transition (s_2, a, μ_2) such that $\mu_1 \; \mathcal{L}(\mathcal{R}_k, k^{-c}) \; \mu_2$. In order to preserve polynomial bounds, we require that also the length of the weak transition is bounded. There are several ways to define such bounds and they are proposed in [21].

We conclude this section with the statement of the main compositionality result for sPASs, which is the goal of the definition presented in this paper.

Theorem 1. *Let* $\mathcal{A}^1$ *and* $\mathcal{A}^2$ *be two families of automata. For each context* $\mathcal{C}$ *compatible with both* $\mathcal{A}^1$ *and* $\mathcal{A}^2$, *if* $\mathcal{A}^1 \lesssim_s \mathcal{A}^2$ *then* $\mathcal{A}^1 || \mathcal{C} \lesssim_s \mathcal{A}^2 || \mathcal{C}$.

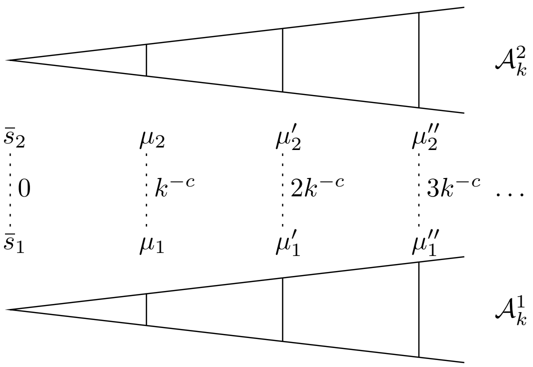


Fig. 1. Graphical representation of the Execution Correspondence Theorem

3.2 Execution Correspondence

The existence of a sPAS from $\mathcal{A}^1$ to $\mathcal{A}^2$ implies a strong correspondence between their probabilistic executions. The formal result is known in concurrency theory as the execution correspondence theorem [17], is known within cryptography in a restrictive form as the mapping lemma [13], and is the basis for reasoning about properties that relate concrete and abstract systems. Informally, the execution correspondence theorem for sPASs states that for every probabilistic execution of polynomial length of the concrete system there exists a corresponding probabilistic execution of the abstract system up to a negligible error.

Definition 6. *Given two families of automata $\mathcal{A}^1$ and $\mathcal{A}^2$ and a family $\mathcal{R}$ of relations from states of $\mathcal{A}^1$ to states of $\mathcal{A}^2$, we say that $\mathcal{A}^1$ is* execution-related by *$\mathcal{R}$ to $\mathcal{A}^2$, denoted by $\mathcal{A}^1$ Ex($\mathcal{R}$) $\mathcal{A}^2$, if for each $c \in \mathbb{N}$, $p \in Poly$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$ and each scheduler σ_1 for $\mathcal{A}^1_k$, if μ_1 is the probability measure induced by σ_1 after n steps, $n \leqslant p(k)$, then there exists a scheduler σ_2 for $\mathcal{A}^2_k$ that reaches, after n steps, a probability measure μ_2 such that $\mu_1 \; \mathcal{L}(\mathcal{R}_k, nk^{-c}) \; \mu_2$.*

We say alternatively that $\mathcal{R}$ is an execution relation *from $\mathcal{A}^1$ to $\mathcal{A}^2$.*

Figure 1 represents graphically the execution correspondence theorem. Given a probabilistic execution of automaton $\mathcal{A}^1_k$, let $\mu_1, \mu'_1, \ldots$ be the measures reached respectively after $1, 2, \ldots$ steps. Then we can find a probabilistic execution of $\mathcal{A}^2_k$, with measures after $1, 2, \ldots$ being $\mu_2, \mu'_2, \ldots$, respectively, such that the matching error grows linearly in the number of steps in the value k^{-c}, which can be made arbitrarily small by increasing the value of the security parameter k.

Theorem 2 (The Execution Correspondence). *If $\mathcal{R}$ is a state polynomially accurate simulation from $\mathcal{A}^1$ to $\mathcal{A}^2$, then $\mathcal{A}^1$ Ex($\mathcal{R}$) $\mathcal{A}^2$.*

Proof (outline). The proof is a classical inductive argument on the number of steps: the base case follows directly from the condition on start states, while the

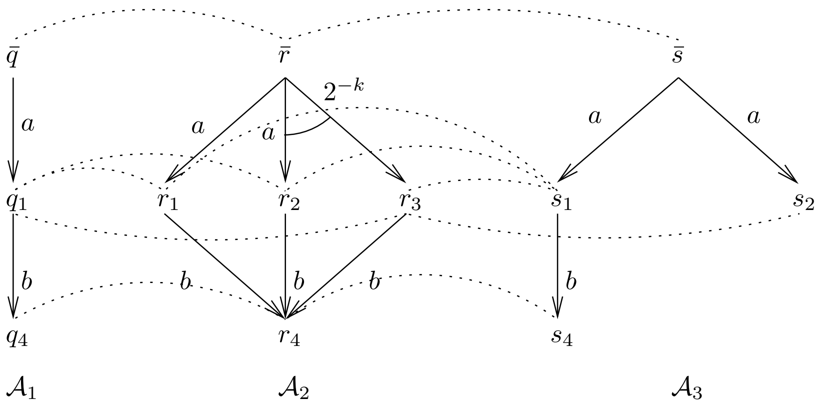


Fig. 2. Naive composition of sPAS does not lead to sPAS

inductive step is based on the following result: if $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg\ St(\mathcal{R}, \varepsilon)\ s_2\} < \varepsilon$, and performing a step from μ_1 we reach measure φ_1, then there exists φ_2 such that $\varphi_1\ \mathcal{L}(\mathcal{R}, \varepsilon + \gamma)\ \varphi_2$ and φ_2 is reached performing a step from μ_2, that is, the following step adds an extra error that is bounded by ε (recall the discussion about the ε-η-approximate simulation). □

The execution-relationship is transitive.

Proposition 1. *Let $\mathcal{A}^1, \mathcal{A}^2, \mathcal{A}^3$ be three families of probabilistic automata and $\mathcal{R}^1, \mathcal{R}^2$ be two relations such that $\mathcal{A}^1\ Ex(\mathcal{R}^1)\ \mathcal{A}^2$ and $\mathcal{A}^2\ Ex(\mathcal{R}^2)\ \mathcal{A}^3$.*
Then, $\mathcal{A}^1\ Ex(\mathcal{R}^1 \circ \mathcal{R}^2)\ \mathcal{A}^3$.

Proof (outline). The proof follows directly from the definition of execution relation: fix $c \in \mathbb{N}$ and $p \in Poly$. $\mathcal{A}^1\ Ex(\mathcal{R}^1)\ \mathcal{A}^2$ implies that for each $c' \in \mathbb{N}$ and each probability measure μ_1 reached within $n \leqslant p(k)$ steps in $\mathcal{A}^1_k$ there exists a probability measure μ_2 reached within n steps in $\mathcal{A}^2_k$ such that $\mu_1\ \mathcal{L}(\mathcal{R}^1_k, nk^{-c'})\ \mu_2$. Similarly, $\mathcal{A}^2\ Ex(\mathcal{R}^2)\ \mathcal{A}^3$ implies that there exists a probability measure μ_3 reached within n steps in $\mathcal{A}^3_k$ such that $\mu_2\ \mathcal{L}(\mathcal{R}^2_k, nk^{-c'})\ \mu_3$. Fix c' to be $2c + 1$. Then we use the following property of ε-lifting: $\rho_1\ \mathcal{L}(\mathcal{R}_1, \varepsilon_1)\ \rho_2\ \mathcal{L}(\mathcal{R}_2, \varepsilon_2)\ \rho_3$ implies $\rho_1\ \mathcal{L}(\mathcal{R}_1 \circ \mathcal{R}_2, \varepsilon_1 + \varepsilon_2)\ \rho_3$ to conclude $\mu_1\ \mathcal{L}(\mathcal{R}^1_k \circ \mathcal{R}^2_k, 2nk^{-c'})\ \mu_3$ which implies $\mu_1\ \mathcal{L}(\mathcal{R}^1_k \circ \mathcal{R}^2_k, nk^{-c})\ \mu_3$ since $2k^{-2c-1} \leqslant k^{-c}$ for each $k > 1$. □

The proposition above is the key ingredient for hierarchical analysis since given a chain of sPAS-related automata each probabilistic execution of the first automaton corresponds to a probabilistic execution of the last automaton up to a negligible error. It would be even nicer to show that sPAS composes; however, this is not the case. Consider the three automata of Figure 2: $\mathcal{A}_1 \lesssim_s \mathcal{A}_2$ and $\mathcal{A}_2 \lesssim_s \mathcal{A}_3$ but the naive composition of sPAS relations (depicted by dotted arcs) leads to a relation that does not satisfy conditions of sPAS. In fact, q_1 is related to s_2 (through r_3) but the transition (q_1, b, δ_{q_4}) can not be matched from s_2, thus the step condition can not be satisfied for δ_{q_1} and δ_{s_2}.

$\mathcal{NG}_k^h(\mathbb{A})$

Signature:

Input:
$\mathit{get\_nonce}(A)$, $A \in \mathbb{A}$

Output:
$\mathit{ret\_nonce}(A, n)$, $n \in \{0,1\}^k$, $A \in \mathbb{A}$

State:

$\mathit{value}_A \in \{0,1\}^k \cup \{\bot\}$, initially $\bot$, $A \in \mathbb{A}$
$\mathit{is\_fresh}_A \in \{T, F, \bot\}$, initially $\bot$, $A \in \mathbb{A}$
$\mathit{frnonces} \subseteq \{0,1\}^k$, initially $\{0,1\}^k$

Transitions:

Input $\mathit{get\_nonce}(A)$
Effect:
$\mathit{value}_A := v$ where $v \in_R \{0,1\}^k$
$$\mathit{is\_fresh}_A := \begin{cases} T & \text{if } v \in \mathit{frnonces} \\ F & \text{otherwise} \end{cases}$$
$\mathit{frnonces} := \mathit{frnonces} \setminus \{v\}$

Output $\mathit{ret\_nonce}(A, n)$
Precondition:
$n = \mathit{value}_A$
Effect:
$\mathit{value}_A := \bot$
$\mathit{is\_fresh}_A := \bot$

Fig. 3. The Nonce Generator $\mathcal{NG}_k^h(\mathbb{A})$

4 Conditional Automata

Consider a public key encryption box that satisfies the *indistinguishability under chosen ciphertext attack* (IND-CCA) property. The box generates random keys and encrypts/decrypts messages. One property of IND-CCA cryptosystems is that repeated keys or repeated ciphertexts are generated with negligible probability. Consider now a more complex system that interacts with the cryptographic box. Can we assume, while analyzing the complex system, that the cryptographic box never generates any repeated key/ciphertext? Conditional automata, together with the compositionality properties of state polynomially accurate simulations, provide us with a positive answer.

We illustrate conditional automata with yet a simpler example. Consider a system that uses nonces generated by a nonce generator that simply draws random numbers of k bits, where k is the security parameter. Can we assume safely that all nonces are different? Again, the answer is positive.

To model this nonce generator, we define an automaton $\mathcal{NG}_k(\mathbb{A})$, where $\mathbb{A}$ is a set of agents, with actions to get and return nonces, respectively. In particular, each time an agent A requires a nonce, the automaton generates a number of k bits that is returned to A. Then we extend this automaton to an automaton $\mathcal{NG}_k^h(\mathbb{A})$ by adding some bookkeeping variables that simply keep track of the past and permits it to determine whether any nonce is repeated. We follow the approach based on history variables [2], for which it is known already [12] that $\mathcal{A} \preceq \mathcal{A}^h$ whenever $\mathcal{A}^h$ is obtained from $\mathcal{A}$ by adding history variables.

Figure 3 depicts $\mathcal{NG}_k^h(\mathbb{A})$. Variables *frnonces* and $\mathit{is\_fresh}_A$ are history variables. The former keeps all values that are not yet chosen as nonces; the latter, one for each agent A, takes value F if the chosen nonce is repeated.

Our next step is to remove from $\mathcal{NG}_k^h(\mathbb{A})$ all those behaviors that lead to repeated nonces. We call the resulting automaton the ideal nonce generator. For the purpose, let G be the set of states of $\mathcal{NG}_k^h(\mathbb{A})$ where no nonce is repeated. We modify the transition relation of $\mathcal{NG}_k^h(\mathbb{A})$ by imposing that states outside G can not be reached by any transition. More precisely, we replace the target measure of each transition by the same measure conditional on G.

Definition 7 (Conditional Automaton). *For a PA $\mathcal{A} = (S, \bar{s}, A, D)$ and $G \subseteq S$, define the G-*conditional *of $\mathcal{A}$, denoted by $\mathcal{A}|G$, to be the PA $\mathcal{A}' = (S, \bar{s}, A, D')$ where $D' = \{(s, a, \mu|G) \mid (s, a, \mu) \in D, \mu(G) > 0\}$.*

At this point we may believe that in $\mathcal{A}|G$ no state outside G is reachable. This is indeed the case provided that the start state of $\mathcal{A}$ is in G.

Proposition 2. *Given a PA $\mathcal{A}$ and $G \subseteq S$. If $\bar{s} \in G$, then all reachable states of $\mathcal{A}|G$ are in G.*

Returning to the nonce generator example, it is immediate to verify that the automaton $\mathcal{NG}_k^h(\mathbb{A})|G$, where G is the set of states where no variable is_fresh_A has value F, is the automaton where the assignment to variable $value_A$ in action $get\_nonce(A)$ is replaced by $value_A := v$ where $v \in_R frnonces$. The last piece that is still missing from the picture is a result that allows us to conclude quickly that the ideal nonce generator is a safe abstraction of the nonce generator that can be used for further analysis.

Indeed, we turn now to our main result, namely, that the conditional automaton construction is sound for sPASs whenever we rule out states that are reachable with negligible probability on computations of polynomial length. We need some preliminary definitions.

For a set F of finite execution fragments, let $Cones(F)$ denote the set $\cup_{\alpha \in F} C_\alpha$. For a set of states B let $\Diamond(B)$ be the set of finite executions whose last state is in B, and let $\Diamond_l(B)$ be the set of executions of $\Diamond(B)$ of length at most l. Thus $Cones(\Diamond(B))$ denotes the event of reaching a state from B, while $Cones(\Diamond_l(B))$ denotes the event of reaching a state from B within l steps.

Definition 8. *Given a probabilistic automaton $\mathcal{A}$ and a set of states B, we say that B is* reachable with probability less than p in $\mathcal{A}$ *if* $\sup_\sigma\{\nu_\sigma(Cones(\Diamond(B)))\} < p$ *and that B is* reachable with probability less than p within l steps in $\mathcal{A}$ *if* $\sup_\sigma\{\nu_\sigma(Cones(\Diamond_l(B)))\} < p$.

Definition 9. *For a family $\mathcal{A}$ of probabilistic automata and a family B of states, we say that B is* polynomially reachable with negligible probability *in $\mathcal{A}$ (or alternatively that B is* negligible *in $\mathcal{A}$) if and only if for each $c \in \mathbb{N}$, each $p \in Poly$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$ the probability to reach states of B_k within $p(k)$ steps in $\mathcal{A}_k$ is less than k^{-c}.*

Remark 7. The phrase "for each $c \in \mathbb{N}$, each $p \in Poly$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$ the probability to reach states of B_k within $p(k)$ steps in $\mathcal{A}_k$ is less than k^{-c}" may appear to be complex; however it is a standard way to say that something is negligible (see discussion about nonces in Section 2.2).

Theorem 3 (Conditional Automaton Theorem). *Let $\mathcal{A}$ be a family of probabilistic automata and G be a family of states such that, for each $k \in \mathbb{N}$, $\bar{s}_k \in G_k$. For each $k \in \mathbb{N}$ let B_k be the set $S_k \setminus G_k$.*

Then the family B is negligible in $\mathcal{A}$ if and only if the family of identity relations is a state polynomially accurate simulation from $\mathcal{A}$ to $\mathcal{A}|G$.

Proof (outline). Negligibility of B imposes a bound on the error between each probability measure μ and $\mu|G$; the execution correspondence theorem bounds the probability to reach states in B (not reachable in $\mathcal{A}|G$) to be negligible. □

An immediate consequence of the conditional automaton theorem is indeed that the ideal nonce generator is a safe abstraction of the nonce generator.

Proposition 3. *Let B_k be the set of states of $\mathcal{NG}^h_k(\mathbb{A})$ where some variable is_fresh$_A$ is F. Let G_k be the set $S_k \setminus B_k$. Then $\mathcal{NG}^h(\mathbb{A}) \lesssim_s \mathcal{NG}^h(\mathbb{A})|G$.*

Proof (outline). Within $p(k)$ steps at most $p(k)$ nonces can be removed from *frnonces*, thus Property 1 implies that the set of states B is negligible in $\mathcal{NG}^h(\mathbb{A})$, hence $\mathcal{NG}^h(\mathbb{A}) \lesssim_s \mathcal{NG}^h(\mathbb{A})|G$. □

The exercise above can be repeated for other cryptographic primitives like encryption and digital signatures, where events like repeated keys or repeated ciphertexts are ruled out. Then, by compositionality, any complex protocol can be analyzed with the ideal versions of the underlying primitives. We do this in [21] where we reprove in a modular way the Dolev-Yao soundness result of [9].

5 Conclusion

In this paper we have introduced Conditional Automata as a tool to abstract security protocols by removing negligible events. We have provided two main theorems: the Conditional Automaton Theorem that relates the negligibility of some states to the existence of a state polynomially accurate simulation between an automaton and its G-conditional counterpart, and the Execution Correspondence Theorem that relates executions of a real protocol with executions of an idealized protocol. We have illustrated our construction via a simple example based on nonce generators; more elaborated examples are available in [21].

The conditional automaton construction turns out to be very simple, as well as the statement of the conditional automaton theorem. All the difficulties lie in setting up an appropriate underlying framework that supports the theorem and inherits the important features of concurrency theory.

We believe that one of the main advantages of our result is the ability to combine in a unique framework proofs based on cryptographic assumptions with proof techniques that have been used extensively and successfully in contexts where there are several concurrent agents. Furthermore, the current framework allows us to work in "hybrid" models where some primitives are described at a concrete level, while other primitives are abstracted a la Dolev-Yao. We are planning to apply these techniques to other more elaborate case studies, as well as investigating the relationship of polynomially accurate simulations with approximated language inclusion and metrics for probabilistic systems.

References

1. Abadi, M., Gordon, A.G.: A calculus for cryptographic protocols: the spi calculus. Information and Computation 148(1), 1–70 (1999)

2. Abadi, M., Lamport, L.: The existence of refinement mappings. Theoretical Computer Science 82(2), 253–284 (1991)
3. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). In: IFIP TCS. LNCS, vol. 2000, pp. 3–22. Springer, Heidelberg (2001)
4. Backes, M., Pfitzmann, B., Waidner, M.: A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015 (2003)
5. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145 (2001)
6. Canetti, R., Cheung, L., Kaynar, D., Liskov, M., Lynch, N.A., Pereira, O., Segala, R.: Using Probabilistic I/O Automata to analyze an oblivious transfer protocol. Tech. Rep. 2005/452, Cryptology ePrint Archive (2005)
7. Canetti, R., Cheung, L., Kaynar, D., Liskov, M., Lynch, N.A., Pereira, O., Segala, R.: Time-bounded task-pIOAs: A framework for analyzing security protocols. In: Dolev, S. (ed.) DISC 2006. LNCS, vol. 4167, pp. 238–253. Springer, Heidelberg (2006)
8. Chatzikokolakis, K., Palamidessi, C.: Making random choices invisible to the scheduler. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 42–58. Springer, Heidelberg (2007)
9. Cortier, V., Warinschi, B.: Computationally sound, automated proofs for security protocols. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 157–171. Springer, Heidelberg (2005)
10. Kantorovich, L.V.: On the translocation of masses. Doklady Akademii Nauk SSSR 37(7-8), 227–229 (1942)
11. Lynch, N.A., Tuttle, M.R.: Hierarchical correctness proofs for distributed algorithms. In: PODC 1987, pp. 137–151 (1987)
12. Lynch, N.A., Vaandrager, F.W.: Forward and backward simulations for timing-based systems. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 397–446. Springer, Heidelberg (1992)
13. Micciancio, D., Warinschi, B.: Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. Journal of Computer Security 12(1), 99–129 (2004)
14. Mitchell, J.C., Ramanathan, A., Scedrov, A., Teague, V.: A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. Theoretical Computer Science 353(1), 118–164 (2006)
15. Mitra, S., Lynch, N.A.: Approximate simulations for task-structured Probabilistic I/O Automata. In: PAuL 2006 (2006)
16. Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: SP 2001, pp. 184–200 (2001)
17. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. Ph.D. thesis, MIT (1995)
18. Segala, R.: Probability and nondeterminism in operational models of concurrency. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 64–78. Springer, Heidelberg (2006)
19. Segala, R., Turrini, A.: Approximated computationally bounded simulation relations for probabilistic automata. In: 20th CSF, pp. 140–154 (2007)
20. Shoup, V.: Sequences of games: A tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004)
21. Turrini, A.: Hierarchical and Compositional Verification of Cryptographic Protocols. Ph.D. thesis, University of Verona (2009), http://www.univr.it/main?ent=catalogoaol&id=337415&page=dettaglioPubblicazione

Representations of Petri Net Interactions

Paweł Sobociński

ECS, University of Southampton, UK

Abstract. We introduce a novel compositional algebra of Petri nets, as well as a stateful extension of the calculus of connectors. These two formalisms are shown to have the same expressive power.

1 Introduction

In part owing to their intuitive graphical representation, Petri nets [28] are often used both in theoretical and applied research to specify systems and visualise their behaviour. On the other hand, process algebras are built around the principle of compositionality: their semantics is given *structurally* so that the behaviour of the whole system is a function of the behaviour of its subsystems. Indeed, Petri nets and process calculi differ in how their underlying semantics is defined: Petri nets via some kind of globally defined transition system of "firing" transitions, and process calculi via an inductively generated (SOS [27]) labelled transition system. As a consequence, the two are associated with different modelling methodologies and reasoning techniques.

There has been much research concentrating on relating the two domains. This paper continues this tradition by showing that a certain class of Petri nets has, in a precise way, the same expressive power as a process calculus.

Technically, we introduce a compositional extension of Condition/Event nets with consume/produce loops. A net is associated with left and right interfaces to which its transitions may connect. Composition of two such nets along a common boundary occurs via a kind of synchronisation of transitions. This notion of compositionality is related to the concept of open nets [4–6].

On the other hand, the process calculus can be considered an extension of (an SOS presentation of) stateless connectors [9] with a very simple notion of state: essentially a one-place buffer. A related extension was considered in [3].

The operations of well-known process algebras have influenced research on Petri nets and various translations have been considered. In the 1990s there was a considerable amount of research that, roughly speaking, related and adapted the operations of the CCS [23] and related calculi to Petri nets. An example of this is the Petri Box calculus [7, 20] and, to a lesser extent, the combinators of Nielsen, Priese and Sassone [26]. More recently, Cerone [11] defined several translations from C/E nets to the Circal process algebra, that like CCS is based on a binary composition and hiding operators. Other recent related work has included endowing Petri nets with labelled transition systems, using techniques and intuitions originating from process calculi, see [21, 24, 29].

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 554–568, 2010.

Conversely, there has also been considerable work on translating process calculi to Petri nets: representative examples include [10, 12, 14, 31]. Recently [15] suggests a set of operations for open nets to which an SOS semantics is assigned.

The operations of the calculus presented in this paper are fundamentally different to those utilised in the aforementioned literature. Indeed, they are closer in nature to those of tile logic [13] and Span(Graph) [18] than to the operations of CCS. More recently, similar operations have been used by Reo [2], glue for component-based systems [8] and the wire calculus [30]. Indeed, in [17] Span(Graph) is used to capture the state space of P/T nets; this work is close in spirit to the translation from nets to terms given in this paper.

Different representations of the same concept can sometimes serve as an indication of its canonicity. Kleene's theorem [19, 22] is a well-known example: on the one hand graphical structures with a globally defined semantics (finite automata) are shown to have the same expressive power as a language with an inductively-defined semantics (regular expressions).

Structure of the paper. Nets with boundaries are introduced in §2 and the relevant process calculus, for the purposes of this paper dubbed the "Petri calculus", is introduced in §3. The translation from nets to process calculus terms is given in §4. A reverse translation is given in §5. Future work is discussed in §6.

2 Nets

Definition 1. For the purposes of this paper a Petri net is a 4-tuple $N = (P, T, {}^\circ{-}, {-}^\circ)$ where[1]:

- P is a set of *places*;
- T is a set of *transitions*;
- ${}^\circ{-}, {-}^\circ \colon T \to 2^P$ are functions.

N is *finite* when both P and T are finite sets.

The obvious notion of net homomorphisms $f\colon N \to M$ is a pair of functions $f_T\colon T_N \to T_M$, $f_P\colon P_N \to P_M$ such that ${}^\circ{-}_N \,;\, 2^{f_P} = f_T \,;\, {}^\circ{-}_M$ and ${-}^\circ_N \,;\, 2^{f_P} = f_T \,;\, {-}^\circ_M$, where $2^{f_P}(X) = \bigcup_{x\in X}\{f_P(x)\}$. For a transition $t \in T$, ${}^\circ t$ and t° are called, respectively, its *pre-* and *post-sets.* Notice that Definition 1 allows transitions with empty pre- and post-sets; this option, while counterintuitive for ordinary nets, will be necessary for nets with boundaries, introduced in §2.1.

Transitions t, u are *independent* when ${}^\circ t \cap {}^\circ u = \varnothing$ and $t^\circ \cap u^\circ = \varnothing$. Note that this notion of independence is quite liberal and allows so-called contact situations. Moreover, a place p can be both in ${}^\circ t$ and t° for some transition t; some authors refer to this as a consume/produce loop; the notion of *contextual net* [25] is related. A set U of transitions is mutually independent when, for all $t, u \in U$, if $t \neq u$ then t and u are independent. Given a set of transitions U let ${}^\circ U = \bigcup_{u\in U} {}^\circ u$ and $U^\circ = \bigcup_{u\in U} u^\circ$.

[1] In the context of C/E nets some authors call places *conditions* and transitions *events*.

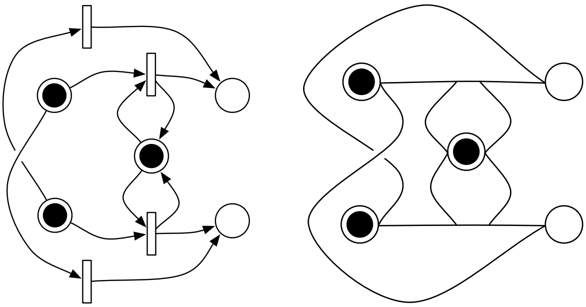


Fig. 1. Traditional and alternative graphical representations of a net

Definition 2 (Semantics). *Let $N = (P, T, {}^\circ-, -^\circ)$ be a net, $X, Y \subseteq P$ and $t \in T$. Write:*

$$(N, X) \to_{\{t\}} (N, Y) \quad \stackrel{\text{def}}{=} \quad {}^\circ t \subseteq X,\ t^\circ \subseteq Y \ \&\ X \backslash {}^\circ t = Y \backslash t^\circ.$$

For $U \subseteq T$ a set of mutually independent transitions, write:

$$(N, X) \to_U (N, Y) \quad \stackrel{\text{def}}{=} \quad {}^\circ U \subseteq X,\ U^\circ \subseteq Y \ \&\ X \backslash {}^\circ U = Y \backslash U^\circ.$$

Note that, for any $X \subseteq P$, $(N, X) \xrightarrow{\varnothing} (N, X)$. States of this transition system will be referred to as markings *of N.*

The left diagram in Fig. 1 demonstrates the traditional graphical representation of a (marked) net. Places are circles; a marking is represented by the presence or absence of tokens. Each transition $t \in T$ is a rectangle; there are directed edges from each place in ${}^\circ t$ to t and from t to each place in t°. This graphical language is a particular way of drawing hypergraphs; the right diagram in Fig. 1 exemplifies another graphical representation, more suitable for representing the notion of nets introduced in this paper. Places are again circles, but each has exactly two *ports*: one on the left and one on the right. Transitions are undirected *links*—each link can connect to any number of ports. Connecting t to the right port p signifies that $p \in {}^\circ t$, connecting t to the left port means that $p \in t^\circ$. Variants of link graphs have been used to characterise various free monoidal categories: see for instance [1, 16].

2.1 Nets with Boundaries

Let $\underline{k}$, $\underline{l}, \underline{m}$, $\underline{n}$ range over finite ordinals: $\underline{n} \stackrel{\text{def}}{=} \{0, 1, \ldots, n-1\}$.

Definition 3. *Let $m, n \in \mathbb{N}$. A (finite)* net with boundaries *$N \colon m \to n$, is a sextuple $(P, T, {}^\circ-, -^\circ, {}^\bullet-, -^\bullet)$ where:*

- $(P, T, {}^\circ-, -^\circ)$ *is a finite net;*
- ${}^\bullet- \colon T \to 2^{\underline{m}}$, $-^\bullet \colon T \to 2^{\underline{n}}$ *are functions.*

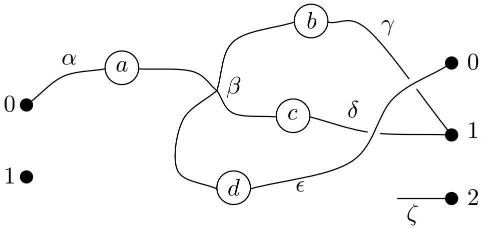


Fig. 2. Representation of a net with boundaries 2 → 3. Here $T = \{\alpha, \beta, \gamma, \delta, \epsilon, \zeta\}$ and $P = \{a, b, c, d\}$. The non-empty values of $^\circ-$ and $-^\circ$ are: $\alpha^\circ = \{a\}$, $^\circ\beta = \{a\}$, $\beta^\circ = \{b, c, d\}$, $^\circ\gamma = \{b\}$, $^\circ\delta = \{c\}$. The non-empty values of $^\bullet-$ and $-^\bullet$ are: $^\bullet\alpha = \{0\}$, $\gamma^\bullet = \{1\}$, $\delta^\bullet = \{1\}$, $\zeta^\bullet = \{2\}$.

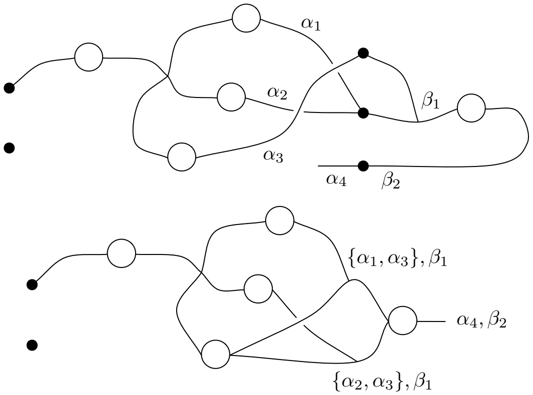


Fig. 3. Illustration of composition of two nets with boundaries

We refer to m and n as, respectively, the left and right boundaries *of N. An example is pictured in Fig. 2.*

Henceforward we shall usually refer to nets with boundaries as simply nets.

The obvious notion of homomorphism between two nets with equal boundaries extends that of ordinary nets: given nets $N, M\colon m \to n$, $f\colon N \to M$ is a pair of functions $f_T\colon T_N \to T_M$, $f_P\colon P_N \to P_M$ such that $^\circ-_N \,;\, 2^{f_P} = f_T \,;\, ^\circ-_M$, $-^\circ_N \,;\, 2^{f_P} = f_T \,;\, -^\circ_M$, $^\bullet-_N = f_T \,;\, ^\bullet-_M$ and $-^\bullet_N = f_T \,;\, -^\bullet_M$. A homomorphism is an isomorphism iff its two components are bijections; we write $N \cong M$ when there is an isomorphism from N to M.

The notion of independence of transitions extends to nets with boundaries in the obvious way: $t, u \in T$ are said to be *independent* when

$$^\circ t \cap {}^\circ u = \varnothing, \quad t^\circ \cap u^\circ = \varnothing, \quad {}^\bullet t \cap {}^\bullet u = \varnothing \quad \text{and} \quad t^\bullet \cap u^\bullet = \varnothing.$$

Let $M\colon l \to m$ and $N\colon m \to n$ be nets. In order to define the composition along their shared boundary, we must first introduce the concept of *synchronisation*: a pair (U, V), with $U \subseteq T_M$ and $V \subseteq T_N$ mutually independent sets of transitions such that:

- $\ast$ $U \cup V \neq \varnothing$;
- $\ast$ $U^\bullet = {}^\bullet V$.

The set of synchronisations inherits an ordering from the subset relation, ie $(U', V') \subseteq (U, V)$ when $U' \subseteq U$ and $V' \subseteq V$. A synchronisation is said to be *minimal* when it is minimal with respect to this order. Let

$$T_{M;N} \overset{\text{def}}{=} \{(U, V) \mid U \subseteq T_M, V \subseteq T_N, (U, V) \text{ a minimal synchronisation}\}$$

Notice that any transition in M or N not connected to the shared boundary m is a minimal synchronisation in the above sense. Define[2] ${}^\circ-, -^\circ : T_{M;N} \to 2^{P_M+P_N}$ by letting ${}^\circ(U, V) = {}^\circ U \cup {}^\circ V$, $(U, V)^\circ = U^\circ \cup V^\circ$. Define ${}^\bullet- : T_{M;N} \to 2^{\underline{l}}$ by ${}^\bullet(U, V) = {}^\bullet U$ and $-^\bullet : T_{M;N} \to 2^{\underline{n}}$ by $(U, V)^\bullet = V^\bullet$. The *composition* of M and N, written $M; N \colon l \to n$, has:

- $\ast$ $T_{M;N}$ as its set of transitions;
- $\ast$ $P_M + P_N$ as its set of places;
- $\ast$ ${}^\circ-, -^\circ \colon T_{M;N} \to 2^{P_M+P_N}$, ${}^\bullet- \colon T_{M;N} \to 2^{\underline{l}}$, $-^\bullet \colon T_{M;N} \to 2^{\underline{n}}$ as above.

An example of a composition of two nets is illustrated in Fig. 3.

Proposition 4

(i) Let $M, M' \colon k \to n$ and $N, N' \colon n \to m$ be nets with $M \cong M'$ and $N \cong N'$. Then $M \mathbin{;} N \cong M' \mathbin{;} N'$

(ii) Let $L \colon k \to l$, $M \colon l \to m$, $N \colon m \to n$ be nets. Then $(L \mathbin{;} M) \mathbin{;} N \cong L \mathbin{;} (M \mathbin{;} N)$ □

We need to define one other binary operation on nets. Given nets $M \colon k \to l$ and $N \colon m \to n$, their *tensor product* is, intuitively the net that results from putting the two nets side-by-side. Concretely, $M \otimes N \colon k + m \to l + n$ has:

- $\ast$ set of transitions $T_M + T_N$;
- $\ast$ set of places $P_M + P_N$;
- $\ast$ ${}^\circ-, -^\circ, {}^\bullet-, -^\bullet$ defined in the obvious way.

2.2 Semantics

Throughout this paper we use two-labelled transition systems. Labels are words in $\{0, 1\}^*$ and are ranged over by α, β. Write $\#\alpha$ for the length of a word α. The intuitive idea is that a transition $p \xrightarrow[\beta]{\alpha} q$ signifies that a system in state p can, in a single step, synchronise with α on its left boundary, β on it right boundary and change its internal state to q.

Definition 5 (Transitions). For $k, l \in \mathbb{N}$, a (k, l)-*transition* is a two-labelled transition of the form $\xrightarrow[\beta]{\alpha}$ where $\alpha, \beta \in \{0, 1\}^*$, $\#\alpha = k$ and $\#\beta = l$. A (k, l) labelled transition system ($(k, l)-$LTS) is a transition system that consists of (k, l)-transitions.

[2] We use $+$ to denote disjoint union.

Definition 6 (Bisimilarity). A simulation on a $(k,\, l)-$LTS is a relation S on its set of states that satisfies the following: if $(v, w) \in S$ and $v \xrightarrow[\beta]{\alpha} v'$ then $\exists w'$ s.t. $w \xrightarrow[\beta]{\alpha} w'$ and $(v', w') \in S$. A bisimulation is a relation S where both S and S^{-1} are simulations. Bisimilarity is the largest bisimulation relation.

For any $k \in \mathbb{N}$, there is a bijection $\ulcorner - \urcorner : 2^k \to \{0,1\}^k$ with

$$\ulcorner U \urcorner_i \overset{\text{def}}{=} \begin{cases} 1 & \text{if } i \in U \\ 0 & \text{otherwise} \end{cases} .$$

Definition 7 (Semantics). *Let $N : m \to n$ be a net and $X, Y \subseteq P_N$. Write:*

$$\begin{aligned} (N,\, X) \xrightarrow[\beta]{\alpha} (N, Y) \quad &\overset{\text{def}}{=} \quad \exists \textit{ mutually independent } U \subseteq T_N \textit{ s.t.} \\ &\qquad (N, X) \to_U (N, Y),\ \alpha = \ulcorner {}^\bullet U \urcorner \ \&\ \beta = \ulcorner U^\bullet \urcorner \end{aligned} \tag{1}$$

Notice that $(N, X) \xrightarrow[0^n]{0^m} (N, X)$.

We conclude this section with a brief remark on the relationship between nets with boundaries and open nets [4, 6]. While open nets are based on P/T nets, a similar construction can be carried out for the variant of net given by Definition 1. Composition in open nets is based on a pushout construction in a category of open-net morphisms. It is not difficult to show that this open net composition can be captured by a composition of nets with boundaries. We omit the details here.

3 Petri Calculus

Here we give the syntax and the structural operational semantics of a simple process calculus, which, for the purposes of this paper, we shall refer to as the *Petri calculus*. It results, roughly, from adding a one-place buffer to the calculus of stateless connectors [9]. The syntax does not feature any binding nor primitives for recursion.

$$P \ ::= \ \bigcirc \mid \odot \mid \mathsf{I} \mid \mathsf{X} \mid \Delta \mid \nabla \mid \bot \mid \top \mid \wedge \mid \vee \mid \downarrow \mid \uparrow \mid P \otimes P \mid P \,;\, P$$

There is an associated sorting. Sorts are of the form $(k,\, l)$, where $k, l \in \mathbb{N}$. The inference rules are given in Fig. 4. Due to their simplicity, a simple induction confirms uniqueness of sorting: if $\vdash P : (k,\, l)$ and $\vdash P : (k',\, l')$ then $k = k'$ and $l = l'$. We shall only consider sortable terms.

Structural inference rules for operational semantics are given in Fig. 5. The rule (Refl) guarantees that any term is always capable of "doing nothing"; note that this is the only rule that applies to $\downarrow$ and $\uparrow$. Each of the rules $(\wedge a)$ and $(\vee a)$ actually represent two rules, one for $a = 0$ and one for $a = 1$.

Bisimilarity on the transition system obtained via the inference rules in Fig. 5 is a congruence. This is important, because it allows us to replace subterms with bisimilar subterms without affecting the behaviour of the overall term. This fact will be relied upon in several proofs.

$$\frac{}{\vdash \bigcirc : (1,1)} \qquad \frac{}{\vdash \odot : (1,1)} \qquad \frac{}{\vdash \mathsf{I} : (1,1)} \qquad \frac{}{\vdash \mathsf{X} : (2,2)}$$

$$\frac{}{\vdash \Delta : (1,2)} \qquad \frac{}{\vdash \nabla : (2,1)} \qquad \frac{}{\vdash \bot : (1,0)} \qquad \frac{}{\vdash \top : (0,1)}$$

$$\frac{}{\vdash \wedge : (1,2)} \qquad \frac{}{\vdash \vee : (2,1)} \qquad \frac{}{\vdash \downarrow : (1,0)} \qquad \frac{}{\vdash \uparrow : (0,1)}$$

$$\frac{\vdash P : (k,l) \quad \vdash R : (m,n)}{\vdash P \otimes R : (k{+}m, l{+}n)} \qquad \frac{\vdash P : (k,n) \quad \vdash R : (n,l)}{\vdash P ; R : (k,l)}$$

Fig. 4. Sort inference rules

$$\frac{}{\bigcirc \xrightarrow[0]{1} \odot}\,(\textsc{TkI}) \qquad \frac{}{\odot \xrightarrow[1]{0} \bigcirc}\,(\textsc{TkO1}) \qquad \frac{}{\odot \xrightarrow[1]{1} \odot}\,(\textsc{TkO2}) \qquad \frac{}{\mathsf{I} \xrightarrow[1]{1} \mathsf{I}}\,(\textsc{Id}) \qquad \frac{a,b\in\{0,1\}}{\mathsf{X} \xrightarrow[ba]{ab} \mathsf{X}}\,(\textsc{Tw})$$

$$\frac{}{\Delta \xrightarrow[11]{1} \Delta}\,(\Delta) \qquad \frac{}{\nabla \xrightarrow[1]{11} \nabla}\,(\nabla) \qquad \frac{}{\bot \xrightarrow{1} \bot}\,(\bot) \qquad \frac{}{\top \xrightarrow[1]{} \top}\,(\top) \qquad \frac{(a\in\{0,1\})}{\wedge \xrightarrow[(1-a)a]{1} \wedge}\,(\wedge a) \qquad \frac{(a\in\{0,1\})}{\vee \xrightarrow[1]{(1-a)a} \vee}\,(\vee a)$$

$$\frac{P \xrightarrow[\mathbf{c}]{\mathbf{a}} Q \quad R \xrightarrow[\mathbf{b}]{\mathbf{c}} S}{P;R \xrightarrow[\mathbf{b}]{\mathbf{a}} Q;S}\,(\textsc{Cut}) \qquad \frac{P \xrightarrow[\mathbf{b}]{\mathbf{a}} Q \quad R \xrightarrow[\mathbf{d}]{\mathbf{c}} S}{P\otimes R \xrightarrow[\mathbf{bd}]{\mathbf{ac}} Q\otimes S}\,(\textsc{Ten}) \qquad \frac{P{:}(k,l)}{P \xrightarrow[0^l]{0^k} P}\,(\textsc{Refl})$$

Fig. 5. Structural rules for operational semantics

Proposition 8. *If $P \sim P'$ then, for any R:*

(i) $(P \,;\, R) \sim (P' \,;\, R)$;
(ii) $(R \,;\, P) \sim (R \,;\, P)$;
(iii) $(P \otimes R) \sim (P' \otimes R)$;
(iv) $(R \otimes P) \sim (R \otimes P')$. □

A *process* is a bisimulation equivalence class of a term. We write $[t] : (m, n)$ for the process that contains $t : (m, n)$.

3.1 Circuit Diagrams

In subsequent sections it will often be convenient to use a graphical language for terms in the Petri calculus. Diagrams in the language will be referred to as *circuit diagrams*. We shall be careful, when drawing diagrams, to make sure that each diagram can be converted to a syntactic expression by "scanning" the diagram from left to right. The following result justifies the usage.

Lemma 9

(i) Let $P : (k, l)$, $Q : (l, m)$, $R : (m, n)$. Then

$$(P \,;\, Q) \,;\, R \sim P \,;\, (Q \,;\, R);$$

(ii) Let $P : (k, l)$, $Q : (m, n)$, $R : (t, u)$. Then

$$(P \otimes Q) \otimes R \sim P \otimes (Q \otimes R);$$

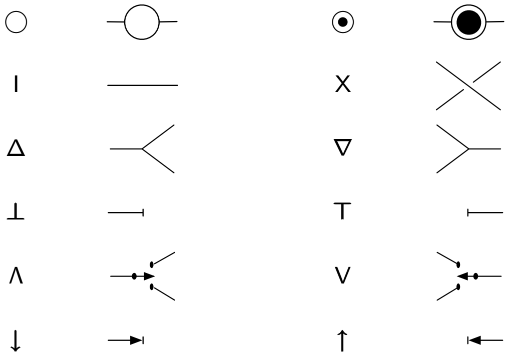


Fig. 6. Circuit diagram components

(iii) Let $P : (k, l)$, $Q : (l, m)$, $R : (n, t)$, $S : (t, u)$. Then

$$(P \mathbin{;} Q) \otimes (R \mathbin{;} S) \sim (P \otimes R) \mathbin{;} (Q \otimes S).$$

Proof. Straightforward, using the inductive presentation of the operational semantics. □

Each of the language constants is represented by a circuit component listed in Fig. 6. For the translation of §4 we need to construct four additional kinds of compound terms, for each $n > 0$:

$$\mathsf{I}_n : (n,\, n) \quad d_n : (0,\, 2n) \quad e_n : (2n,\, 0) \quad \mathit{\Delta}_n : (n,\, 2n) \quad \nabla_n : (2n,\, n)$$

with operational semantics characterised by:

$$\frac{\alpha\in\{0,1\}^n}{\mathsf{I}_n \xrightarrow[\alpha]{\alpha} \mathsf{I}_n} \quad \frac{\alpha\in\{0,1\}^n}{d_n \xrightarrow[\alpha\alpha]{} d_n} \quad \frac{\alpha\in\{0,1\}^n}{e_n \xrightarrow{\alpha\alpha} e_n} \quad \frac{\alpha\in\{0,1\}^n}{\mathit{\Delta}_n \xrightarrow[\alpha\alpha]{\alpha} \mathit{\Delta}_n} \quad \frac{\alpha\in\{0,1\}^n}{\nabla_n \xrightarrow[\alpha]{\alpha\alpha} \nabla_n} \tag{2}$$

First, $\mathsf{I}_n = \bigotimes_n \mathsf{I}$. Now because d_n and e_n, as well as $\mathit{\Delta}_n$ and ∇_n are symmetric, here we only construct d_n and $\mathit{\Delta}_n$. Each is defined recursively:

$$d_1 = \top \mathbin{;} \Delta \qquad d_{n+1} = d_n \mathbin{;} (\mathsf{I}_n \otimes d_1 \otimes \mathsf{I}_n); (\mathsf{I}_{n+1} \otimes \mathsf{X}_n)$$

$$\mathit{\Delta}_1 = \Delta \qquad \mathit{\Delta}_{n+1} = (\Delta \otimes \mathit{\Delta}_n) \mathbin{;} (\mathsf{I} \otimes \mathsf{X}_n \otimes \mathsf{I}_n)$$

where also $\mathsf{X}_n : (n+1,\, n+1)$ is defined recursively:

$$\mathsf{X}_1 = \mathsf{X} \qquad \mathsf{X}_{n+1} = (\mathsf{X}_n \otimes \mathsf{I}) \mathbin{;} (\mathsf{I}_n \otimes \mathsf{X}).$$

An easy induction on the derivation of a transition confirms that these construction produce terms whose semantics is characterised by (2).

3.2 Relational Forms

For $\theta \in \{\mathsf{X}, \Delta, \nabla, \bot, \top, \wedge, \vee, \downarrow, \uparrow\}$ let T_θ denote the set of terms generated by the following grammar:

$$T_\theta \ ::= \ \theta \mid \mathsf{I} \mid T_\theta \otimes T_\theta \mid T_\theta \mathbin{;} T_\theta.$$

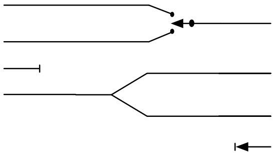


Fig. 7. Right relational form of $f \colon \underline{4} \to 2^{\underline{4}}$ defined $f(0), f(1) = \{0\}$, $f(2) = \varnothing$ and $f(3) = \{1, 2\}$

We shall use t_θ to range over terms of T_θ. We now identify two classes of terms of the Petri calculus: the *relational forms.*

Definition 10. A term $t : (k,\, l)$ is in *right relational form* when

$$t = t_{\bot} \; ; t_{\Delta} \; ; t_{\mathsf{X}} \; ; t_{\mathsf{V}} \; ; t_{\uparrow}.$$

Dually, t is said to be in *left relational form* when

$$t = t_{\downarrow} \; ; t_{\wedge} \; ; t_{\mathsf{X}} \; ; t_{\nabla} ; t_{\top}.$$

The following result spells out the significance of the relational forms.

Lemma 11. *For each function $f \colon \underline{k} \to 2^{\underline{l}}$ there exists a term $\rho_f : (k,\, l)$ in right relational form, the dynamics of which are characterised by the following:*

$$\frac{}{\rho_f \xrightarrow[\ulcorner V \urcorner]{\ulcorner U \urcorner} \rho_f} \;\Leftrightarrow\; U \subseteq \underline{k} \textit{ s. t. } \forall u, v \in U.\; u \neq v \Rightarrow f(u) \cap f(v) = \varnothing \;\&\; V = f(U)$$

The symmetric result holds for functions $f \colon \underline{k} \to 2^{\underline{l}}$ and terms $t : (l,\, k)$ in left relational form. Write $\lambda_f : (l,\, k)$ for any term in left relational form that corresponds to f in the above sense.

Proof. Any function $f : \underline{k} \to 2^{\underline{l}}$ induces a triple $(\underline{m}, l_f \colon \underline{m} \to \underline{k}, r_f \colon \underline{m} \to \underline{l})$ where l_f and r_f are jointly injective, ie the function $(l_f, r_f) \colon \underline{m} \to \underline{k} \times \underline{l}$ is injective, and $f(i) = \bigcup_{j \in l_f^{-1}(i)} r_f(j)$ where $l_f^{-1}(i) = \{j \mid l_f(j) = i\}$. Any two such triples are isomorphic as spans of functions. It is not difficult to verify that any function $l_f \colon \underline{m} \to \underline{k}$ gives rise to a term t_{l_f} of the form $t_{\bot} \; ; t_{\Delta} \; ; t_{\mathsf{X}}$, the semantics of which are characterised by $t_{l_f} \xrightarrow[\ulcorner l_f^{-1}(U) \urcorner]{\ulcorner U \urcorner} t_{l_f}$ for any $U \in \underline{k}$ where for all $u, v \in U$, $l_f^{-1}(u) \cap l_f^{-1}(v) = \varnothing$. Also, any function $r_f \colon \underline{m} \to \underline{l}$ gives rise to a term t_{r_f} of the form $t_{\mathsf{X}} \; ; t_{\mathsf{V}} \; ; t_{\downarrow}$, the semantics of which are $t_{r_f} \xrightarrow[\ulcorner W \urcorner]{\ulcorner V \urcorner} t_{r_f}$ where $\forall w \in W$ there exists unique $v \in V$ such that $r_f(v) = w$. It thus suffices to let $\rho_f = t_{l_f} \; ; t_{r_f}$. □

A simple example is given in Fig. 7. Note that not all terms $t : (k,\, l)$ in right relational form are bisimilar to ρ_f for some $f \colon \underline{k} \to 2^{\underline{l}}$; a simple counterexample is $\Delta \; ; \mathsf{V} : (1,\, 1)$.

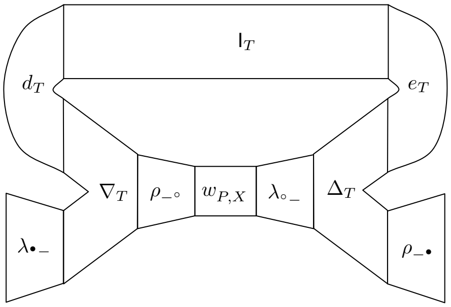


Fig. 8. Diagrammatic representation of the translation from a marked net to a term.

4 Translating Nets to Petri Calculus Terms

Here we present a translation from nets with boundaries, defined in §2, to the process calculus defined in §3. Let $N\colon m \to n = (P, T, {}^\circ-, -^\circ, {}^\bullet-, -^\bullet)$ be a finite net with boundary and $X \subseteq P$ a marking. Assume, without loss of generality, that $P = \underline{p}$ and $T = \underline{t}$ for some $p, t \in \mathbb{N}$. Let

$$w_{P,\,X} : (p,\, p) \overset{\text{def}}{=} \bigotimes_{i<p} m_i \quad \text{where} \quad m_i \overset{\text{def}}{=} \begin{cases} \odot & \text{if } i \in X \\ \bigcirc & \text{otherwise} \end{cases}$$

The following technical result will be useful for showing that the encoding of this section is correct.

Lemma 12. *$w_{P,X} \xrightarrow[\ulcorner W \urcorner]{\ulcorner Z \urcorner} Q$ iff $Q = w_{P,Y}$, $W \subseteq X$, $Z \subseteq Y$ and $X \backslash W = Y \backslash Z$.*

Proof. Examination of rules (Tki), (Tko1) and (TKO2), together with the rule (Ten). □

The translation of N can now be expressed as:

$$T_{N,X} \overset{\text{def}}{=} (d_T \otimes \lambda_{\bullet-}) \;;\; (\mathsf{I}_T \otimes (\nabla_T \;;\; \rho_{-\circ} \;;\; w_{P,X} \;;\; \lambda_{\circ-} \;;\; \Delta_T)); (e_T \otimes \rho_{-\bullet}).$$

A circuit diagram representation of the above term is illustrated in Fig. 8.

The encoding preserves and reflects semantics in a very tight manner, as shown by the following.

Theorem 13. *Let N be a finite net. The following hold:*

(i) if $(N, X) \xrightarrow[\beta]{\alpha} (N, Y)$ then $T_{N,X} \xrightarrow[\beta]{\alpha} T_{N,Y}$;
(ii) conversely, if $T_{N,X} \xrightarrow[\beta]{\alpha} Q$ then there exists Y such that $Q = T_{N,Y}$ and $(N, X) \xrightarrow[\beta]{\alpha} (N, Y)$.

Proof. (i) If $(N, X) \xrightarrow[\beta]{\alpha} (N, Y)$ then there exists a set $U \subseteq \underline{t}$ of mutually independent transitions such that $(N, X) \to_U (N, Y)$, with $\alpha = \ulcorner {}^\bullet U \urcorner$ and $\beta = \ulcorner U^\bullet \urcorner$. Using the conclusion of Lemma 12, we have

$$w_{P,X} \xrightarrow[\ulcorner {}^\circ U \urcorner]{\ulcorner U^\circ \urcorner} w_{P,Y}.$$

Now, using the conclusion of Lemma 11 and (Cut) we obtain transition

$$\rho_{-\circ} \;;\; w_{P,X} \;;\; \lambda_{\circ -} \xrightarrow[\ulcorner U \urcorner]{\ulcorner U \urcorner} \rho_{-\circ} \;;\; w_{P,Y} \;;\; \lambda_{\circ -}$$

and subsequently

$$\nabla_T \;;\; \rho_{-\circ} \;;\; w_{P,X} \;;\; \lambda_{\circ -} \;;\; \Delta_T \xrightarrow[\ulcorner U \urcorner \ulcorner U \urcorner]{\ulcorner U \urcorner \ulcorner U \urcorner} \nabla_T \;;\; \rho_{-\circ} \;;\; w_{P,Y} \;;\; \lambda_{\circ -} \;;\; \Delta_T$$

Certainly $\mathsf{I}_T \xrightarrow[\ulcorner U \urcorner]{\ulcorner U \urcorner} \mathsf{I}_T$, thus using the semantics of d_T and e_T we obtain:

$$T_{N,X} \xrightarrow[\ulcorner U^{\bullet} \urcorner]{\ulcorner {}^{\bullet}U \urcorner} T_{N,Y}$$

as required.
(ii) If $T_{N,X} \xrightarrow[\beta]{\alpha} Q$ then $Q = (d_T \otimes \lambda_{\bullet -}) \;;\; Q_1 \;;\; (e_T \otimes \rho_{-\bullet})$ and

$$I_T \otimes (\nabla_T \;;\; \rho_{-\circ} \;;\; w_{P,X} \;;\; \lambda_{\circ -} \;;\; \Delta_T) \xrightarrow[\ulcorner U' \urcorner \ulcorner U' \urcorner \ulcorner V' \urcorner]{\ulcorner U \urcorner \ulcorner U \urcorner \ulcorner V \urcorner} Q_1$$

For some $U, V, U', V' \subseteq \underline{t}$ with $\alpha = \ulcorner {}^{\bullet}V \urcorner$ and $\beta = \ulcorner V'^{\bullet} \urcorner$. The structure of (Ten) and the semantics of I_T imply that $U = U'$ and $Q_1 = \mathsf{I}_T \otimes Q_2$ with

$$\nabla_T \;;\; \rho_{-\circ} \;;\; w_{P,X} \;;\; \lambda_{\circ -} \;;\; \Delta_T \xrightarrow[\ulcorner U \urcorner \ulcorner V' \urcorner]{\ulcorner U \urcorner \ulcorner V \urcorner} Q_2$$

Now the semantics of Δ_T implies that $U = V$ and conversely, the semantics of ∇_T that $U = V'$, moreover $Q_2 = \nabla_T \;;\; Q_3 \;;\; \delta_T$ with

$$\rho_{-\circ} \;;\; w_{P,X} \;;\; \lambda_{\circ -} \xrightarrow[\ulcorner U \urcorner]{\ulcorner U \urcorner} Q_3$$

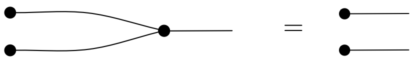
Finally, using the conclusion of Lemma 11, we obtain $Q_3 = \rho_{-\circ} \;;\; Q_4 \;;\; \lambda_{\circ -}$ and

$$w_{P,X} \xrightarrow[\ulcorner {}^{\circ}U \urcorner]{\ulcorner U^{\circ} \urcorner} Q_4$$

In particular, we obtain that $Q_4 = w_{P,Y}$ and $(N, X) \xrightarrow[\beta]{\alpha} (N, Y)$. □

5 Translating Petri Calculus Terms to Nets

Each of the constants of the Petri calculus has a corresponding net with the same semantics: this translation is given in Fig. 9. The naive way of extending this translation to all terms would then be to let $[\![t_1 \;;\; t_2]\!] = [\![t_1]\!] \;;\; [\![t_2]\!]$ and $[\![t_1 \otimes t_2]\!] = [\![t_1]\!] \otimes [\![t_2]\!]$. The naive translation does not reflect behaviour, essentially because of three problematic compositions that involve $\mathsf{\Lambda}$ and/or V. First, consider the net that would result from translating the term $\mathsf{V} \;;\; \bot : (2,\, 0)$:



According to the inductive system in Fig. 5, the non-trivial transitions of the operational semantics of $\mathsf{V} \;;\; \bot$ are: $\mathsf{V} \;;\; \bot \xrightarrow{10} \mathsf{V} \;;\; \bot$ and $\mathsf{V} \;;\; \bot \xrightarrow{01} \mathsf{V} \;;\; \bot$. Now

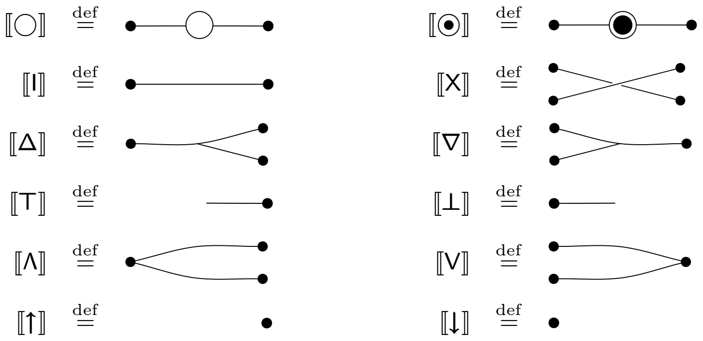
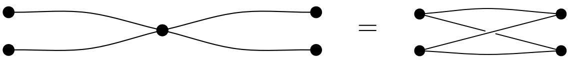


Fig. 9. Translation from calculus constants to nets with marking

$[\![\vee ; \bot]\!]$ has the above transitions, but also an extra transition: $[\![\vee ; \bot]\!] \xrightarrow{11} [\![\vee ; \bot]\!]$. The second problematic composition is $\top ; \wedge$, which is symmetric to the above situation.

The third and final problematic composition amongst constants arises when translating the term $\vee ; \wedge : (2, 2)$. Here the net composition of the translated components is:



Now the non-trivial derivable transitions are

$$(\vee ; \wedge) \xrightarrow[01]{01} (\vee ; \wedge),\ (\vee ; \wedge) \xrightarrow[10]{10} (\vee ; \wedge),\ (\vee ; \wedge) \xrightarrow[10]{01} (\vee ; \wedge),\ (\vee ; \wedge) \xrightarrow[01]{10} (\vee ; \wedge).$$

Again, the encoding introduces an additional transition

$$[\![\vee ; \wedge]\!] \xrightarrow[11]{11} [\![\vee ; \wedge]\!].$$

The solution, then, is to first transform each term t into a bisimilar term t' in a form which allows compositional translation into a bisimilar net $N_{t'}$.

The initial transformation is best understood via the circuit diagram representation of a term, the soundness of which is ensured by Lemma 9. We say that a term is in *composable form* when, in its circuit diagram:

(i) any occurrence of $\vee$ is connected on the right to either the right boundary, another occurrence of $\vee$, $\bigcirc$ or $\odot$;
(ii) any occurrence of $\wedge$ is connected on the left to either the left boundary, another occurrence of $\wedge$, $\bigcirc$ or $\odot$;

If a term t can can be transformed into the above form then it follows that it can be written as $t_1 = t_\wedge ; t_2 ; t_\vee$, where in t_2 any occurrence of $\wedge$ and $\vee$ is within a subterm of the form $t_\vee ; \bigcirc ; t_\wedge$ (*), or $t_\vee ; \odot ; t_\wedge$ (**). Terms of the form (*) and (**) translate into correct nets, by a case straightforward analysis, the translation can be continued compositionally to obtain a net N_{t_1} with marking X_{t_1} such that $(N_{t_1}, X_{t_1}) \sim t_1$.

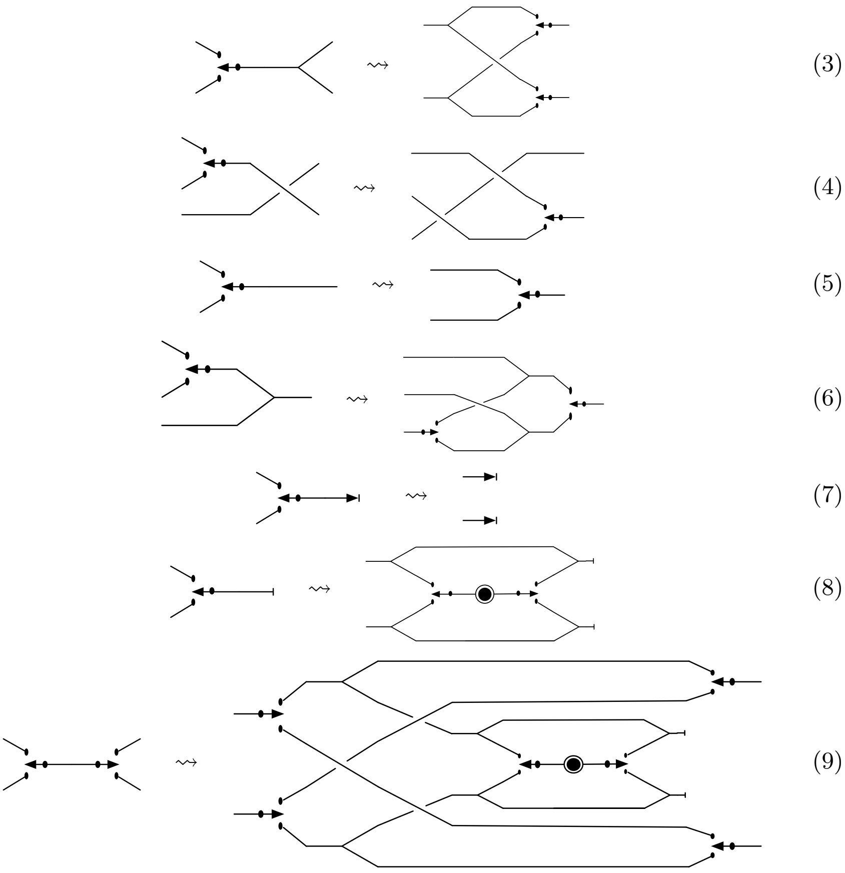


Fig. 10. Rewriting system for $\vee$

Theorem 14. *For each term t there exists a net N_t such that $t \sim N_t$.*

Proof. By the above reasoning, it suffices to show that a term can be transformed into composable form. For this we apply transformations to individual occurrences of $\vee$ and $\wedge$ until the requirements of composable form are met. The rules for $\vee$ are given in Fig. 10. Rules (8) and (9) deal with $\vee$'s problematic compositions. The other rules "push $\vee$ to the right". The complete rewriting system is obtained by including the symmetric versions of (3), (4), (5), (6), (7) and (8) for $\wedge$. □

6 Conclusion and Future Work

We showed that the class of nets with boundaries has the same expressiveness as a simple process calculus with operations that are fundamentally different from those of CCS, but closely related to operations of coordination languages. As future work it will be interesting to capture the expressive power of other classes of nets, for instance P/T nets with boundaries, with extensions of the process calculus presented here.

Acknowledgment. The author thanks Jennifer Lantair and the anonymous referees for helpful suggestions.

References

1. Abramsky, S.: Abstract scalars, loops and free traced and strongly compact closed categories. In: Fiadeiro, J.L., Harman, N.A., Roggenbach, M., Rutten, J. (eds.) CALCO 2005. LNCS, vol. 3629, pp. 1–29. Springer, Heidelberg (2005)
2. Arbab, F.: Reo: a channel-based coordination model for component composition. Math. Struct. Comp. Sci. 14(3), 1–38 (2004)
3. Arbab, F., Bruni, R., Clarke, D., Lanese, I., Montanari, U.: Tiles for Reo. In: WADT 2008. LNCS, vol. 5486. Springer, Heidelberg (2008)
4. Baldan, P., Corradini, A., Ehrig, H., Heckel, R.: Compositional modelling of reactive systems using open nets. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 502–518. Springer, Heidelberg (2001)
5. Baldan, P., Corradini, A., Ehrig, H., Heckel, R.: Compositional semantics for open Petri nets based on deterministic processes. Math. Struct. Comp. Sci. 15(1), 1–35 (2005)
6. Baldan, P., Corradini, A., Ehrig, H., Heckel, R., König, B.: Bisimilarity and behaviour-preserving reconfigurations of Petri nets. Log. Meth. Comput. Sci. 4(4), 1–41 (2008)
7. Best, E., Devillers, R., Hall, J.G.: The box calculus: A new causal algebra with multi-labelled communication. In: Rozenberg, G. (ed.) APN 1992. LNCS, vol. 609, pp. 21–69. Springer, Heidelberg (1992)
8. Bliudze, S., Sifakis, J.: A notion of glue expressiveness for component-based systems. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 508–522. Springer, Heidelberg (2008)
9. Bruni, R., Lanese, I., Montanari, U.: A basic algebra of stateless connectors. Theor. Comput. Sci. 366, 98–120 (2006)
10. Busi, N., Gorrieri, R.: A Petri net semantics for the π-calculus. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 145–159. Springer, Heidelberg (1995)
11. Cerone, A.: Implementing Condition/Event nets in the circal process algebra. In: Kutsche, R.-D., Weber, H. (eds.) FASE 2002. LNCS, vol. 2306, pp. 49–63. Springer, Heidelberg (2002)
12. Degano, P., Nicola, R.D., Montanari, U.: A distributed operational semantics for CCS based on C/E systems. Acta Inform. 26, 59–91 (1988)
13. Gadducci, F., Montanari, U.: The tile model. In: Proof, Language and Interaction: Essays in Honour of Robin Milner, pp. 133–166. MIT Press, Cambridge (2000)
14. Goltz, U.: CCS and Petri nets. In: Guessarian, I. (ed.) LITP 1990. LNCS, vol. 469. Springer, Heidelberg (1990)
15. Groote, J.F., Voorhoeve, M.: Operational semantics for Petri net components. Theor. Comput. Sci. 379(1-2), 1–19 (2007)
16. Joyal, A., Street, R.: The geometry of tensor calculus, I. Adv. Math. 88, 55–112 (1991)

17. Katis, P., Sabadini, N., Walters, R.F.C.: Representing P/T nets in Span(Graph). In: Johnson, M. (ed.) AMAST 1997. LNCS, vol. 1349, pp. 307–321. Springer, Heidelberg (1997)
18. Katis, P., Sabadini, N., Walters, R.F.C.: Span(Graph): an algebra of transition systems. In: Johnson, M. (ed.) AMAST 1997. LNCS, vol. 1349, pp. 322–336. Springer, Heidelberg (1997)
19. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Automata Studies, pp. 3–41. Princeton University Press, Princeton (1956)
20. Koutny, M., Esparza, J., Best, E.: Operational semantics for the Petri box calculus. In: Jonsson, B., Parrow, J. (eds.) CONCUR 1994. LNCS, vol. 836, pp. 210–225. Springer, Heidelberg (1994)
21. Leifer, J.J., Milner, R.: Transition systems, link graphs and Petri nets. Math. Struct. Comp. Sci. 16(6), 989–1047 (2006)
22. McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata. IEEE Trans. Electronic Computers 9, 39–47 (1960)
23. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
24. 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)
25. Montanari, U., Rossi, F.: Contextual nets. Acta Inform. 32(6), 545–596 (1995)
26. Nielsen, M., Priese, L., Sassone, V.: Characterizing behavioural congruences for Petri nets. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 175–189. Springer, Heidelberg (1995)
27. Plotkin, G.D.: A structural approach to operational semantics. J. Logic Algebr. Progr. 60-61, 17–139 (2004); Originally appeared as Technical Report DAIMI FN-19, University of Aarhus (1981)
28. Reisig, W.: Petri nets: an introduction. EATCS Monographs on Theoretical Computer Science. Springer, Heidelberg (1985)
29. Sassone, V., Sobociński, P.: A congruence for Petri nets. In: Petri Nets and Graph Transformation (PNGT 2004). ENTCS, vol. 127, pp. 107–120 (2005)
30. Sobociński, P.: A non-interleaving process calculus for multi-party synchronisation. In: Interaction and Concurrency (ICE 2009). EPTCS, vol. 12 (2009)
31. 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 1987. LNCS, vol. 259. Springer, Heidelberg (1987)

Communicating Transactions[⋆]
(Extended Abstract)

Edsko de Vries, Vasileios Koutavas, and Matthew Hennessy

Trinity College Dublin
{Edsko.de.Vries,Vasileios.Koutavas,Matthew.Hennessy}@cs.tcd.ie

Dedicated to Robin Milner

Abstract. We propose a novel language construct called *communicating transactions*, obtained by dropping the isolation requirement from classical transactions, which can be used to model automatic error recovery in distributed systems. We extend CCS with this construct and give a simple semantics for the extended calculus, called TransCCS. We develop a behavioural theory which is sound and complete with respect to the may-testing preorder, and use it to prove interesting laws and reason compositionally about example systems. Finally, we prove that communicating transactions do not increase the observational power of processes; thus CCS equivalences are preserved in the extended language.

1 Introduction

Distributed systems such as web services [7] consist of a number of autonomous nodes in a network that communicate through message passing. As web services are increasingly designed by combining other web services through so-called *mashup* technologies [2], the complexity of these systems grows and *error recovery* becomes ever more difficult.

The usefulness of the transaction concept for the treatment of errors in such a setting has been recognized by both academia [19,5] and industry [10,12]. Error recovery in such transactions is based on *compensation*: services must programmatically bring the system back to a consistent state when an error has happened. In a distributed system of many independent components this may be difficult and error prone.

In many situations, however, automatic error recovery is possible through the use of classical techniques such as *rollback recovery* [16]. Processes store enough local state to be able to roll back after an error, and a rollback in one node may cause other nodes to rollback so that all nodes have a consistent view of the system state. The extent of the rollbacks can be limited through *coordinated checkpointing* [13], where processes coordinate to create a point beyond which they do not need to be rolled back.

In this paper we define a novel language construct of *communicating transactions*, which can be used to model the combination of rollback recovery and coordinated checkpointing. We give a high-level semantics of communicating

[⋆] This research was supported by SFI project SFI 06 IN.1 1898.

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 569–583, 2010.

transactions in a calculus called TransCCS, an extension of CCS, and develop a compositional theory for this calculus based on may-testing equivalence.

Unlike traditional transactions, communicating transactions are not isolated: they may communicate with other processes or transactions in their environment. When a transaction communicates with its environment and subsequently fails, both the transaction and its environment will be rolled back to a consistent state. Transactions can commit to request a checkpoint; when transactions communicate, all must commit before the checkpoint is actually created (*cf.* the standard two-phase commit algorithm used for distributed transactions [27]).

In TransCCS we use $\llbracket P \rhd_k Q \rrbracket$ to denote a transaction named k which currently runs P; a transaction is replaced by its *default* P after a commit, and by its *alternative* Q after an abort. *Restarting* transactions are modelled by recursive transactions $\mu X.\, \llbracket P \rhd_k X \rrbracket$. A transaction can be aborted by the system at any point, and can commit using the language primitive $\mathtt{co}\ k$.

To give an intuition of communicating transactions we consider an informal semantics for communicating transactions. An example idealized execution of a system consisting of a merchant M (left) and a bank B (right) is given by:

$$\begin{array}{llr|ll}
 & & \mathit{req}.\, \llbracket \tau.\overline{\mathit{tr}}.(\mathtt{co}\ k \mid \overline{\mathit{ack}}) \rhd_k \overline{\mathit{err}} \rrbracket & \mu X.\, \llbracket \mathit{tr}.\tau.\mathtt{co}\ l \rhd_l X \rrbracket & \\
\text{(Place order)} & \xrightarrow{\mathit{req}} & \llbracket \tau.\overline{\mathit{tr}}.(\mathtt{co}\ k \mid \overline{\mathit{ack}}) \rhd_k \overline{\mathit{err}} \rrbracket & \mu X.\, \llbracket \mathit{tr}.\tau.\mathtt{co}\ l \rhd_l X \rrbracket & \\
\text{(Process order)} & \xrightarrow{\tau} & \llbracket \overline{\mathit{tr}}.(\mathtt{co}\ k \mid \overline{\mathit{ack}}) \rhd_k \overline{\mathit{err}} \rrbracket & \mu X.\, \llbracket \mathit{tr}.\tau.\mathtt{co}\ l \rhd_l X \rrbracket & \\
\text{(Request transfer)} & \xrightarrow{\tau} & \llbracket (\mathtt{co}\ k \mid \overline{\mathit{ack}}) \rhd_k \overline{\mathit{err}} \rrbracket & \llbracket \tau.\mathtt{co}\ l \rhd_l B \rrbracket & \\
\text{(System aborts } l\text{)} & \xrightarrow{\tau} & & \mu X.\, \llbracket \mathit{tr}.\tau.\mathtt{co}\ l \rhd_l X \rrbracket & (1) \\
\text{(Cascading rollback)} & & \llbracket \overline{\mathit{tr}}.(\mathtt{co}\ k \mid \overline{\mathit{ack}}) \rhd_k \overline{\mathit{err}} \rrbracket & & \\
\text{(Second attempt)} & \xrightarrow{\tau}\xrightarrow{\tau} & \llbracket (\mathtt{co}\ k \mid \overline{\mathit{ack}}) \rhd_k \overline{\mathit{err}} \rrbracket & \llbracket \mathtt{co}\ l \rhd_l B \rrbracket & (2) \\
\text{(Commit)} & \xrightarrow{\tau}\xrightarrow{\tau} & \overline{\mathit{ack}} & \mathbf{0} & (3) \\
\text{(Acknowledge Order)} & \xrightarrow{\overline{\mathit{ack}}} & \mathbf{0} & \mathbf{0} &
\end{array}$$

In this trace M accepts an order on channel *req* and enters transaction k. Inside the transaction, M processes the order and issues a transfer request on *tr* to the bank B, which enters a (restarting) transaction l. The communication on *tr* should be considered tentative as it involves transactions k and l which are still subject to system failure. When the system decides to abort the l transaction in (1), it must also roll back the k transaction to a point before the transfer request in order to maintain global consistency; the k transaction, however, does not need to re-process the order. The second attempt to communicate between the transactions in (2) is also tentative, and only becomes a definitive action in (3) when both transactions have issued their commits. The acknowledgement of the order is then sent on *ack*. If at any point the system decides to abort the merchant transaction k (perhaps due to multiple failures to perform the transfer by the bank), an error signal is sent on *err*.

A direct formalization of this informal semantics would be quite complicated; for example dependencies between the various transactions would have to be maintained dynamically, and some notion of coordinated checkpointing or rollback would need to be implemented. In this paper we show that we can abstract away from such details through a simple concept called *embedding*. Specifically, we make the following contributions.

1. We give a simple reductions semantics for TransCCS by augmenting the standard semantics of CCS with a rule for *embedding* a process into a transaction, and two simple rules for committing and aborting transactions (Sect. 2).
2. We give a compositional behavioural theory for TransCCS (Sect(s). 3 to 5), based on *non prefix-closed* sets of traces derived by a Labelled Transition System (LTS), which is sound and complete with respect to *may-testing* [15]. The theory distinguishes between standard processes such as $a.b.\mathbb{0}$ in which all actions are definitive, and transactions $\llbracket a.b.\mathtt{co}\ k \triangleright_k \mathbb{0} \rrbracket$ where the actions are tentative until transaction k commits.
3. We use the theory to prove a number of interesting laws about communicating transactions, including a theorem that transactions do not increase the observational power of processes and therefore CCS equivalences are preserved in TransCCS (Sect. 5). We also use the theory to reason compositionally about simple distributed systems (Sect. 6).
4. We study an extension to our calculus, TransCCS$^{\mu\mathsf{ab}}$, in which aborts are programmable (Sect. 7). We show that, provided all transactions are restarting, the characterization of may-testing in TransCCS is also valid in TransCCS$^{\mu\mathsf{ab}}$; we prove this through a simple fully abstract translation into TransCCS.

2 TransCCS

The syntax of TransCCS is that of CCS extended with a construct $\llbracket P \triangleright_k Q \rrbracket$, denoting a transaction which is currently running its default P but which will be replaced by its alternative Q when it is aborted, and a construct $\mathtt{co}\ k$ to commit transactions, replacing it by its default.[1] The syntax and the reduction semantics are shown in Fig. 1; as usual a ranges over a set of actions Act on which is defined a bijective function $(\overline{\cdot}) : Act \to Act$, used to formalize communication, and μ ranges over Act_τ, the set Act augmented with a new action τ, used to represent internal activity. We use the standard abbreviations for CCS terms.

Although communication does not cross transaction boundaries, transactions can communicate through (non-deterministic) embedding:[2]

Example 1. Consider the reductions from a system consisting of the transaction $\llbracket a.(\mathtt{co} \mid c) \triangleright_k b \rrbracket$ running in parallel with the simple process $\overline{a}$. Before communication can take place, the process must be embedded into the transaction; by embedding it into both the default and the alternative part of the transaction, we can restore the process to a consistent state after an abort. The possible traces are summarized in the graph below; note that a rollback (through R-Ab) remains possible until the commit has been executed. □

$$\begin{tikzcd}
\llbracket a.(\mathtt{co}\ k \mid c) \triangleright_k b \rrbracket \mid \overline{a} \arrow[r, "\text{R-Emb}"] \arrow[dr, bend right, "\text{R-Ab}"'] & \llbracket a.(\mathtt{co}\ k \mid c) \mid \overline{a} \triangleright_k b \mid \overline{a} \rrbracket \arrow[r, "\text{R-Comm}"] \arrow[d, "\text{R-Ab}"] & \llbracket \mathtt{co}\ k \mid c \triangleright_k b \mid \overline{a} \rrbracket \arrow[r, "\text{R-Co}"] \arrow[dl, bend left, "\text{R-Ab}"] & c \\
& b \mid \overline{a} & &
\end{tikzcd}$$

[1] After the commit, any remaining (possibly prefixed) $\mathtt{co}\ k$ statements behave like $\mathbb{0}$.

[2] Communication-driven embedding results in an equivalent but more complicated semantics.

Syntax

$$\begin{array}{rlll}
P, Q ::= \sum \mu_i.P_i & \text{guarded choice} & \mid \llbracket P \triangleright_k Q \rrbracket & \text{transaction } (k \text{ bound in } P) \\
\mid (P \mid Q) & \text{parallel} & \mid \mathsf{co}\ k & \text{commit} \\
\mid \nu a.P & \text{hiding} & \mid \mu X.P & \text{recursion}
\end{array}$$

Reduction Rules ($\rightarrow$) is the least relation that satisfies

R-Comm
$$\frac{a_i = \overline{a}_j}{\sum_{i \in I} a_i.P_i \mid \sum_{j \in J} a_j.Q_j \rightarrow P_i \mid Q_j}$$

R-Emb
$$\frac{k \notin R}{\llbracket P \triangleright_k Q \rrbracket \mid R \rightarrow \llbracket P \mid R \triangleright_k Q \mid R \rrbracket}$$

R-Tau
$$\frac{\mu_i = \tau}{\sum_{i \in I} \mu_i.P_i \rightarrow P_i}$$

R-Co
$$\frac{}{\llbracket P \mid \mathsf{co}\ k \triangleright_k Q \rrbracket \rightarrow P}$$

R-Ab
$$\frac{}{\llbracket P \triangleright_k Q \rrbracket \rightarrow Q}$$

R-Rec
$$\frac{}{\mu X.P \rightarrow P[X := \mu X.P]}$$

R-Str
$$\frac{P \equiv P' \rightarrow Q' \equiv Q}{P \rightarrow Q}$$

and is closed under the contexts $C ::= [] \mid (C \mid Q) \mid \llbracket C \triangleright_k Q \rrbracket \mid \nu a.C$.
Structural equivalence ($\equiv$) contains the usual rules for parallel and hiding.

Fig. 1. Language Definition

Two or more transactions can communicate by taking mutual embedding steps, which is possible because transactions can be nested arbitrarily in TransCCS.[3]

Example 2. Consider again the Merchant M and Bank B transactions, together with a client $C = \overline{req}.P$. An example trace of $(C \mid M \mid B)$ is given by:

$$\begin{aligned}
& \overline{req}.P \mid req.\llbracket \tau.\overline{tr}.(\mathsf{co}\ k \mid \overline{ack}) \triangleright_k \overline{err} \rrbracket \mid \mu X.\llbracket tr.\tau.\mathsf{co}\ l \triangleright_l X \rrbracket \\
\xrightarrow{\text{R-Comm}}\xrightarrow{\text{R-Tau}}\ & P \mid \llbracket \overline{tr}.(\mathsf{co}\ k \mid \overline{ack}) \triangleright_k \overline{err} \rrbracket \mid \mu X.\llbracket tr.\tau.\mathsf{co}\ l \triangleright_l X \rrbracket \\
\xrightarrow{\text{R-Emb}}\xrightarrow{\text{R-Rec}}\ & P \mid \llbracket \overline{tr}.(\mathsf{co}\ k \mid \overline{ack}) \mid \llbracket tr.\tau.\mathsf{co}\ l \triangleright_l B \rrbracket \triangleright_k \overline{err} \mid B \rrbracket \\
\xrightarrow{\text{R-Emb}}\ & P \mid \llbracket \llbracket \overline{tr}.(\mathsf{co}\ k \mid \overline{ack}) \mid tr.\tau.\mathsf{co}\ l \triangleright_l \overline{tr}.(\mathsf{co}\ k \mid \overline{ack}) \mid B \rrbracket \triangleright_k \ldots \rrbracket \\
\xrightarrow{\text{R-Comm}}\ & P \mid \llbracket \llbracket (\mathsf{co}\ k \mid \overline{ack}) \mid \tau.\mathsf{co}\ l \triangleright_l \overline{tr}.(\mathsf{co}\ k \mid \overline{ack}) \mid B \rrbracket \triangleright_k \overline{err} \mid B \rrbracket \qquad (*) \\
\xrightarrow{\text{R-Co}}\ & P \mid \llbracket (\mathsf{co}\ k \mid \overline{ack}) \triangleright_k \overline{err} \mid B \rrbracket \xrightarrow{\text{R-Co}} P \mid \overline{ack}
\end{aligned}$$

An alternative trace starting from $(*)$ begins with an abort of the bank:

$$\begin{aligned}
& P \mid \llbracket \llbracket (\mathsf{co}\ k \mid \overline{ack}) \mid \tau.\mathsf{co}\ l \triangleright_l \overline{tr}.(\mathsf{co}\ k \mid \overline{ack}) \mid B \rrbracket \triangleright_k \overline{err} \mid B \rrbracket \\
\xrightarrow{\text{R-Ab}}\ & P \mid \llbracket \overline{tr}.(\mathsf{co}\ k \mid \overline{ack}) \mid B \triangleright_k \overline{err} \mid B \rrbracket
\end{aligned}$$

[3] We can embed B into M or vice-versa, but the two embeddings are equivalent.

The application of R-Ab also rolls back the merchant to before the output $\overline{tr}$; this is in conformance with the informal semantics in the introduction, as is the fact that the internal computation of the merchant is not rolled back. □

3 May Testing

We now apply the standard definition of may-testing to TransCCS. We will model a successful outcome of a test by a top-level output on a fresh channel ω.

Definition 1 (Barb). *$P{\Downarrow_a}$ iff there exist P_1 and P_2 such that $P \rightarrow^* P_1 \mid a.P_2$.*

Definition 2 (May-Testing Preorder). *We write $P \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{may}} Q$ iff for all processes T containing a fresh name ω, $(P \mid T){\Downarrow_\omega}$ implies $(Q \mid T){\Downarrow_\omega}$. We write $P \eqsim_{\mathrm{may}} Q$ if $P \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{may}} Q$ and $Q \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{may}} P$.*

Example 3. Consider the systems $P_1 = \llbracket a.b.\mathsf{co}\; k \triangleright_k \mathbf{0} \rrbracket$, $P_2 = a.b$ and the test $T = \overline{a}.\omega$. When applied to P_2 the test succeeds since we reach the state $b \mid \omega$, but when applied to P_1 it fails since this leads to the failed state $\llbracket b.\mathsf{co}\; k \mid \omega \triangleright_k \mathbf{0} \mid \overline{a}.\omega \rrbracket$ (which does not have an ω-barb). Consequently $a.b \mathrel{\underset{\sim}{\not\sqsubset}}_{\mathrm{may}} \llbracket a.b.\mathsf{co}\; k \triangleright_k \mathbf{0} \rrbracket$. □

Our definition of barbs as top-level actions ensures that whenever $P{\Downarrow_\omega}$ then the action ω in P is definitive rather than tentative; the structure of processes in TransCCS ensures that top-level actions do not depend on the commitment of any transaction.[4] This is crucial to our notion of testing; for example the failed state above has the possibility of performing the action ω but this is tentative, as it depends on the transaction k committing. If k is aborted then this apparent success of the test would have to be rolled back.

In Sect. 5 we give a characterization of may-testing equivalence, with which we can give easy proofs for the following laws.

Proposition 1 (Uncommitted actions). *Actions within a transaction are not observable unless the transaction commits. For all P, Q, and R such that $k \notin R$ (in particular, $\mathsf{co}\; k \notin R$), we have*

$$\llbracket R \triangleright_k Q \rrbracket \eqsim_{\mathrm{may}} Q \qquad \mu X.\, \llbracket R \triangleright_k X \rrbracket \eqsim_{\mathrm{may}} \mathbf{0} \tag{1}$$

$$\llbracket P + R \triangleright_k Q \rrbracket \eqsim_{\mathrm{may}} \llbracket P \triangleright_k Q \rrbracket \qquad \mu X.\, \llbracket P + R \triangleright_k X \rrbracket \eqsim_{\mathrm{may}} \mu X.\, \llbracket P \triangleright_k X \rrbracket \tag{2}$$

Proposition 2 (Restarting transactions) $\mu X.\, \llbracket P \triangleright_k X \rrbracket \eqsim_{\mathrm{may}} \llbracket P \triangleright_k \mathbf{0} \rrbracket$

Proposition 3 (Transactions versus processes).

$$\llbracket a.\mathsf{co}\; k \triangleright_k \mathbf{0} \rrbracket \eqsim_{\mathrm{may}} a \qquad \mu X.\, \llbracket a.\mathsf{co}\; k \triangleright_k X \rrbracket \eqsim_{\mathrm{may}} a \tag{3}$$

$$\llbracket P \mid \mathsf{co}\; k \triangleright_k \mathbf{0} \rrbracket \eqsim_{\mathrm{may}} P \qquad \mu X.\, \llbracket P \mid \mathsf{co}\; k \triangleright_k X \rrbracket \eqsim_{\mathrm{may}} P \tag{4}$$

$$\llbracket P \triangleright_k Q \rrbracket \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{may}} \tau.P + \tau.Q \qquad \mu X.\, \llbracket P \triangleright_k X \rrbracket \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{may}} P \tag{5}$$

[4] The theory of biorthogonality [24] yields the same barbs for our reduction semantics.

$$\textsc{L-Act}\ \frac{}{\sum \mu_i.P_i \xrightarrow{\mu_i} P_i} \qquad \textsc{L-Par}\ \frac{\mathcal{P} \xrightarrow{\widetilde{k}(\mu)} \mathcal{P}'}{\mathcal{P} \mid \mathcal{Q} \xrightarrow{\widetilde{k}(\mu)} \mathcal{P}' \mid \mathcal{Q}} \qquad \textsc{L-Trans}\ \frac{\mathcal{P} \xrightarrow{\widetilde{l}(\mu)} \mathcal{P}'}{[\![\mathcal{P} \rhd_k \mathcal{Q}]\!] \xrightarrow{k(\widetilde{l}(\mu))} [\![\mathcal{P}' \rhd_k \mathcal{Q}]\!]}$$

$$\textsc{L-Rec}\ \frac{}{\mu X.\mathcal{P} \xrightarrow{\tau} \mathcal{P}[X := \mu X.\hat{\mathcal{P}}]} \qquad \textsc{L-Hide}\ \frac{\mathcal{P} \xrightarrow{\mu} \mathcal{P}' \qquad a \notin \mu}{\nu a.\mathcal{P} \xrightarrow{\mu} \nu a.\mathcal{P}'} \qquad \textsc{L-Comm}\ \frac{\mathcal{P} \xrightarrow{\widetilde{k}(a)} \mathcal{P}' \qquad \mathcal{Q} \xrightarrow{\widetilde{k}(\overline{a})} \mathcal{Q}'}{\mathcal{P} \mid \mathcal{Q} \xrightarrow{\widetilde{k}(\tau)} \mathcal{P}' \mid \mathcal{Q}'}$$

(eliding L-Trans for secondary transactions)

Fig. 2. LTS: Standard Actions

4 Compositional LTS

4.1 Distributed Transactions

The use of the embedding rule R-Emb in the reduction semantics gives an easy to understand description of the execution of communicating transactions, but prevents compositional reasoning: parallel processes are no longer separate after embedding. For example, when trying to understand why the application of a test T to a system P is successful, embedding makes it difficult to disentangle the contributions made by T and P. After a number of reduction steps components of the test are distributed throughout the system, and vice-versa.

The LTS implements embedding differently. It is defined over an extended language, TransCCS°, where transactions are distributed as a primary transaction, denoted by $[\![\mathcal{P} \rhd_k \mathcal{Q}]\!]$, and zero or more secondary transactions, denoted by $[\![\mathcal{P} \rhd_k \mathcal{Q}]\!]^\circ$. The system from Ex. 1 has the following trace in the LTS:

$$[\![a.(\mathtt{co}\ k \mid c) \rhd_k b]\!] \mid \overline{a} \xrightarrow{\mathtt{emb}\ k} [\![a.(\mathtt{co}\ k \mid c) \rhd_k b]\!] \mid [\![\overline{a} \rhd_k \overline{a}]\!]^\circ$$
$$\xrightarrow{k(\tau)} [\![\mathtt{co}\ k \mid c \rhd_k b]\!] \mid [\![\mathbb{0} \rhd_k \overline{a}]\!]^\circ \xrightarrow{\mathtt{co}\ k} c$$

The application of R-Emb is mimicked by the action $\mathtt{emb}\ k$ in the LTS, and the right process becomes a secondary k-transaction. The parallel composition of the primary transaction and secondary k-transaction should be thought of as modelling the transaction $[\![a.(\mathtt{co}\ k \mid c) \mid \overline{a} \rhd_k b \mid \overline{a}]\!]$. The two processes remain separate, however, allowing for compositional reasoning. A step $\mathcal{P} \xrightarrow{\mathtt{emb}\ k} [\![\mathcal{P} \rhd_k \mathcal{P}]\!]^\circ$ into a secondary k-transaction with no corresponding primary transaction models the embedding of $\mathcal{P}$ into a k transaction which is part of the environment.

The LTS is shown in Fig(s). 2 and 3. Judgements take the form:

- Communication actions $\mathcal{P} \xrightarrow{\widetilde{l}(\mu)} \mathcal{P}'$, which represent the tentative execution of μ inside the transactions $\widetilde{l}$.
- Broadcast actions $\mathcal{P} \xrightarrow{\beta} \mathcal{P}'$ where β can take the forms $(\mathtt{co}\ k)$ for committing, $(\mathtt{ab}\ k)$ for aborting, and $(\mathtt{emb}\ k)$ for embedding.

B-CoPri
$$\frac{\mathcal{P} \equiv \mathcal{P}' \mid \mathtt{co}\ k}{[\![\mathcal{P} \rhd_k \mathcal{Q}]\!] \xrightarrow{\mathtt{co}\ k} \mathcal{P}'}$$

B-CoSec
$$\frac{}{[\![\mathcal{P} \rhd_k \mathcal{Q}]\!]^\circ \xrightarrow{\mathtt{co}\ k} \mathcal{P}}$$

B-Ab
$$\frac{}{[\![\mathcal{P} \rhd_k \mathcal{Q}]\!] \xrightarrow{\mathtt{ab}\ k} \mathcal{Q}}$$

B-Emb
$$\frac{}{\mathcal{P} \xrightarrow{\mathtt{emb}\ k} [\![\mathcal{P} \rhd_k \hat{\mathcal{P}}]\!]^\circ}$$

B-Trans
$$\frac{\mathcal{P} \xrightarrow{\beta} \mathcal{P}' \qquad \beta \neq \mathtt{co}\ k, \mathtt{ab}\ k}{[\![\mathcal{P} \rhd_k \mathcal{Q}]\!] \xrightarrow{\beta} [\![\mathcal{P}' \rhd_k \mathcal{Q}]\!]}$$

B-Par
$$\frac{\mathcal{P} \xrightarrow{\beta} \mathcal{P}' \qquad \mathcal{Q} \xrightarrow{\beta} \mathcal{Q}'}{\mathcal{P} \mid \mathcal{Q} \xrightarrow{\beta} \mathcal{P}' \mid \mathcal{Q}'}$$

B-Rec
$$\frac{}{\mu X.P \xrightarrow{\beta} \mu X.P}$$

B-Act
$$\frac{}{\sum \mu_i.P_i \xrightarrow{\beta} \sum \mu_i.P_i}$$

B-Co
$$\frac{}{\mathtt{co}\ k \xrightarrow{\beta} \mathtt{co}\ k}$$

B-Hide
$$\frac{\mathcal{P} \xrightarrow{\beta} \mathcal{P}'}{\nu a.\mathcal{P} \xrightarrow{\beta} \nu a.\mathcal{P}'}$$

(eliding B-Ab and B-Trans for secondary transactions)

Fig. 3. LTS: Broadcast actions

We will refer to $\widetilde{k}(\tau)$ and broadcast actions as *silent* actions, and likewise to traces containing only silent actions as silent traces.

Communication actions are marked with their enclosing transactions (rule L-Trans). A $k(a)$ action can be matched by a parallel $k(\overline{a})$ action (L-Comm), modelling internal communication within the k-transaction.

When a primary k-transaction is ready to commit (B-CoPri), all secondary k-transactions must follow (B-CoSec). This is achieved by viewing the action $\mathtt{co}\ k$ as a broadcast action, which is propagated throughout the system (B-Par); non-transactions are unaffected by this action. Aborts are handled in a similar manner, although even primary transactions are subject to random system aborts (B-Ab). Embedding (B-Emb) is also a broadcast action to allow the distributed components of a process to be embedded simultaneously. Note that B-Hide does not require $a \notin \beta$ since we cannot restrict transaction names.

The rules in the LTS are subject to an implicit wellformedness condition, formally defined in [26], which guarantees that the distribution of transactions in a term indeed models a single transaction. For example, it prohibits terms such as

$$[\![[\![\mathcal{P}_1 \rhd_k \mathcal{Q}_1]\!] \rhd_l \ldots]\!]^\circ \mid [\![\mathcal{R} \rhd_k \mathcal{R}]\!]^\circ \qquad \text{(illformed)}$$

The k-transaction cannot both be and not be embedded inside the l-transaction, and we therefore reject this term as illformed. Wellformedness also includes some technical but natural conditions that deal with freshness.

To support distribution, transactions are not binders in TransCCS$^\circ$ but are renamed when necessary (B-Emb, L-Rec) using an operation $\hat{\mathcal{P}}$. The implementation of $\hat{\mathcal{P}}$ is unimportant, but it must have the obvious properties (distribution over the constructors of the language, replacing names by sufficiently fresh ones, etc.), and defined so that two components of the same transaction (for instance, a primary and a secondary k-transaction) must be given the *same* new name. We use $\mathrm{tn}\,(\mathcal{P})$ to denote the set of names of the transactions in $\mathcal{P}$.

4.2 Relation to Reduction Semantics

To be able to formalize the relation between the reduction semantics and the LTS, we need to specify the mapping between terms in TransCCS° and TransCCS. We define an operation ($\rightsquigarrow$) which combines two k-transactions in a TransCCS° term into a single k-transaction.

Definition 3 (Merging). *($\rightsquigarrow$) is the least pre-congruence closed under structural equivalence that satisfies*

$$\begin{array}{l} [\![\mathcal{P}_1 \rhd_k \mathcal{Q}_1]\!] \;\;\mid [\![\mathcal{P}_2 \rhd_k \mathcal{Q}_2]\!]^\circ \rightsquigarrow [\![\mathcal{P}_1 \mid \mathcal{P}_2 \rhd_k \mathcal{Q}_1 \mid \mathcal{Q}_2]\!] \\ [\![\mathcal{P}_1 \rhd_k \mathcal{Q}_1]\!]^\circ \mid [\![\mathcal{P}_2 \rhd_k \mathcal{Q}_2]\!]^\circ \rightsquigarrow [\![\mathcal{P}_1 \mid \mathcal{P}_2 \rhd_k \mathcal{Q}_1 \mid \mathcal{Q}_2]\!]^\circ \end{array}$$

We use the symbol ($\leftrightsquigarrow$) for the symmetric closure of ($\rightsquigarrow$).

If we apply $\rightsquigarrow$ repeatedly, we eventually end up with a process with a single term for each transaction. If moreover the original process contained a primary k-transaction for every k (and not just secondary transactions), then we can regard the result as a TransCCS term. We overload $\mathcal{P} \rightsquigarrow P$ to describe this translation from a TransCCS° term to a TransCCS term.

We can now state that the LTS and the reduction semantics coincide:

Theorem 1 (Reduction semantics vs LTS). *Let $\mathcal{P} \rightsquigarrow P$.*

1. *If $P \rightarrow Q$ then there exist a process $\mathcal{Q}$ and silent action μ such that $\mathcal{P} \xrightarrow{\mu} \mathcal{Q}$ and $\mathcal{Q} \rightsquigarrow Q$. Moreover, if $\mu = \mathtt{emb}\ k$ then $k \in \mathrm{tn}\,(\mathcal{P})$.*
2. *If $\mathcal{P} \xrightarrow{t} \mathcal{Q}$, where t is a silent trace, and whenever $\mathtt{emb}\ k \in t$ then $k \in \mathrm{tn}\,(\mathcal{P})$, then there exist Q such that $P \rightarrow^* Q$ and $\mathcal{Q} \rightsquigarrow Q$.*

5 Characterization of May Testing

TransCCS encodes the complex interactions between communicating transactions. In this section we prove that the behaviour of transactional processes with respect to may-testing is characterized by a class of simple traces, which we call *clean traces*. We also prove that a weaker preorder which only uses non-transactional, sequential tests coincides with the may-testing preorder, and therefore CCS equivalences are preserved in TransCCS.

5.1 Clean Traces

Clean traces correspond to traces in the LTS where actions are never rolled back and are committed at the end of the trace: intuitively, every action in a clean trace eventually becomes definitive. Unlike LTS traces, however, clean traces do not include transaction names or broadcast actions. To enforce that all actions become definitive, the formal definition of clean traces (Fig. 4) is parametrized by a finite set of names Δ. Actions within a k-transaction can only occur in a clean trace if $k \in \Delta$ (C-Act, C-Emb), in which case k must commit at the end of the trace (C-Co) and cannot be aborted (C-Ab). We use $\mathtt{co}\ \{k_1, \ldots, k_n\}$ for the process ($\mathtt{co}\ k_1 \mid \cdots \mid \mathtt{co}\ k_n$).

$$\dfrac{\mathcal{P} \xrightarrow{\widetilde{k}(\mu)} \mathcal{P}'' \xrightarrow{t}_{\Delta} \mathcal{P}' \qquad \widetilde{k} \subseteq \Delta}{\mathcal{P} \xrightarrow{\mu,t}_{\Delta} \mathcal{P}'}\,\textsc{C-Act} \qquad \dfrac{\mathcal{P} \xrightarrow{\mathtt{ab}\ k} \mathcal{P}'' \xrightarrow{t}_{\Delta} \mathcal{P}' \qquad k \notin \Delta}{\mathcal{P} \xrightarrow{t}_{\Delta} \mathcal{P}'}\,\textsc{C-Ab}$$

$$\dfrac{\mathcal{P} \xrightarrow{\mathtt{emb}\ k} \mathcal{P}'' \xrightarrow{t}_{\Delta} \mathcal{P}' \qquad k \in \Delta}{\mathcal{P} \xrightarrow{t}_{\Delta} \mathcal{P}'}\,\textsc{C-Emb} \qquad \dfrac{\mathcal{P} \xrightarrow{\mathtt{co}\ \Delta} \mathcal{P}'}{\mathcal{P} \xrightarrow{\epsilon}_{\Delta} \mathcal{P}'}\,\textsc{C-Co}$$

Fig. 4. Clean Traces

Example 4. Let $\mathcal{P} = a.b + c$. As C-Act does not constrain top-level actions, the set of clean traces of $\mathcal{P}$ is $\{\epsilon, a, ab, c\}$ irrespective of the choice of Δ. □

Example 5. Let $\mathcal{P} = [\![a.b.\mathtt{co}\ k \rhd_k c]\!]$. If we choose $\Delta = \{k\}$, we can only derive the clean trace ab:

$$\dfrac{[\![a.b.\mathtt{co}\ k \rhd_k c]\!] \xrightarrow{k(a)} \qquad \dfrac{[\![b.\mathtt{co}\ k \rhd_k c]\!] \xrightarrow{k(b)} \quad \dfrac{[\![\mathtt{co}\ k \rhd_k c]\!] \xrightarrow{\mathtt{co}\ k} \mathbf{0}}{[\![\mathtt{co}\ k \rhd_k c]\!] \xrightarrow{\epsilon}_{\{k\}} \mathbf{0}}\,\textsc{C-Co} \quad k \in \{k\}}{[\![b.\mathtt{co}\ k \rhd_k c]\!] \xrightarrow{b}_{\{k\}} \mathbf{0} \qquad\qquad k \in \{k\}}\,\textsc{C-Act}}{[\![a.b.\mathtt{co}\ k \rhd_k c]\!] \xrightarrow{a,b}_{\{k\}} \mathbf{0}}\,\textsc{C-Act}$$

With this choice of Δ we cannot derive the empty trace because the k transaction is unable to commit immediately (nor can it be aborted). However, if we pick $\Delta = \emptyset$, we can derive the clean traces ϵ (using C-Co) and c (using C-Ab).

The singleton trace a is not derivable as a clean trace with *any* choice of Δ. As in the derivation above, we need $k \in \Delta$ to do a $k(a)$ action but the transaction is unable to commit until the b action. Clean traces are thus not prefix closed: $\mathcal{P}$ cannot do a definite a without also doing a definitive b.

Similarly, the trace abc is not derivable as a clean trace with any choice of Δ, because we need $k \in \Delta$ to do the $k(a)$ and $k(b)$ actions, and $k \notin \Delta$ to abort the k transaction and do the c action. $\mathcal{P}$ can *either* do a definitive a and b, *or* a definitive c, but not both. □

Normally the choice of Δ is not important:

Definition 4. *We write $\mathcal{P} \xrightarrow{t}_{CL}$ iff t is a clean trace of $\mathcal{P}$, that is $\exists \Delta, \mathcal{P}'$ such that $\mathcal{P} \xrightarrow{t}_{\Delta} \mathcal{P}'$. We write $\mathcal{P} \overset{t}{\Rightarrow}_{CL}$ to denote that t is a weak clean trace of $\mathcal{P}$.*[5]

Example 6. The set of clean traces of $[\![a.b.\mathtt{co}\ k \rhd_k c]\!]$ (Ex. 5) is $\{\epsilon, ab, c\}$. □

Example 7. Let $\mathcal{P} = \nu a.\Big([\![a.b.\mathtt{co}\ k \rhd_k \mathbf{0}]\!] \mid [\![\overline{a}.(\tau.\mathtt{co}\ l + \overline{b}) \rhd_l \mathbf{0}]\!]\Big)$. The set of clean traces of $\mathcal{P}$ is $\{\epsilon, \tau b\}$. The trace $\tau\tau$ (internal communication on both a and b) is not derivable for any Δ because the l transaction cannot commit after doing a $\overline{b}$. The trace $\tau\overline{b}$ is not derivable for similar reasons. □

[5] Since clean traces are CCS traces, we can use the standard definition of a weak trace.

5.2 Soundness and Completeness

Our theory of may-testing is based on weak clean trace inclusion.

Definition 5 (Clean Trace Preorder). *We write $\mathcal{P} \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{tr}} \mathcal{Q}$ iff $\mathcal{P} \overset{t}{\Rightarrow}_{CL}$ implies $\mathcal{Q} \overset{t}{\Rightarrow}_{CL}$.*

The clean trace preorder is sound and complete with respect to the may-testing preorder (Def. 2).

Theorem 2 (Soundness). *If $P \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{tr}} Q$ then $P \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{may}} Q$.*

Theorem 3 (Completeness). *If $P \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{may}} Q$ then $P \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{tr}} Q$.*

We now define a weaker testing preorder that uses only non-transactional, sequential tests, which coincides with the may-testing and clean trace preorders.

Definition 6 (Non-Transactional Testing Preorder). *We write $\mathcal{P} \mathrel{\hat{\underset{\sim}{\sqsubset}}}_{\mathrm{may}} \mathcal{Q}$ iff for all tests T of the form $a_1.a_2.\dots.a_n.\omega$, $(P \mid T){\Downarrow_\omega}$ implies $(Q \mid T){\Downarrow_\omega}$.*

Theorem 4 (Conservativity). *$P \mathrel{\hat{\underset{\sim}{\sqsubset}}}_{\mathrm{may}} Q$ iff $P \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{may}} Q$.*

The final theorem entails that equivalent CCS processes are also equivalent TransCCS processes; i.e. communicating transactions do not increase the distinguishing power of the language.

5.3 Proof Outline

The proof that may-testing is characterized by weak clean traces (Thm(s). 2 and 3) is a non-trivial result, and we can give but a sketch of the proof here. A more detailed proof can be found in a companion technical report [26].

For may-testing we are interested in (silent) traces that result in a top-level barb (ω). The first result states that whenever a process can ring with an arbitrary trace, it can ring with a clean trace:

Proposition 4. *Let s be a silent trace such that $\mathcal{P} \xrightarrow{s} \mathcal{R} \mid \omega$. Then there exists Δ, silent clean t, and $\mathcal{R}'$ such that $\mathcal{P} \xrightarrow{t}_{\Delta} \mathcal{R}' \mid \omega$.*

Proof (outline). First we inspect s and pick a Δ containing exactly the transactions that commit in s. Then we construct t by induction on s. Actions in s which happen inside transactions that do not commit (and are not in Δ) cannot contribute to the ring and are simply skipped. Similarly, if a transaction aborts we make sure to abort it before any other action inside the transaction. Finally, we delay all commits to the end of the trace. The final process $\mathcal{R}'$ may differ from the final process after the original trace $\mathcal{R}$, but it will ring. As a simple example, the trace

$$\llbracket \tau.\mathbb{0} \rhd_k \mathbb{0} \rrbracket \mid \llbracket \tau.\mathbb{0} \rhd_l \llbracket \tau.\omega \mid \mathtt{co}\ m \rhd_m \mathbb{0} \rrbracket \rrbracket \xrightarrow{k(\tau),l(\tau),\mathtt{ab}\ l,\mathtt{co}\ m,\tau} \llbracket \mathbb{0} \rhd_k \mathbb{0} \rrbracket \mid \omega$$

will be converted to the trace $\mathtt{ab}\ l, m(\tau), \mathtt{co}\ m$, used to derive the clean trace

$$\llbracket \tau.\mathbb{0} \rhd_k \mathbb{0} \rrbracket \mid \llbracket \tau.\mathbb{0} \rhd_l \llbracket \tau.\omega \mid \mathtt{co}\ m \rhd_m \mathbb{0} \rrbracket \rrbracket \xrightarrow{\tau}_{\{m\}} \llbracket \tau.\mathbb{0} \rhd_k \mathbb{0} \rrbracket \mid \omega \qquad \square$$

We assume the standard definition of "zipping" of two CCS-like traces $(t \# t')$ that allows interleaving and communication between the actions of the traces [26]. The next result is crucial; it states that zipping is a meaningful operation on clean traces; i.e., that the parallel composition of two processes can do any clean trace in the zip of the clean traces of the individual processes.

Proposition 5 (Zipping). *Let* $\mathcal{P} \xrightarrow{t_1}_{\Delta} \mathcal{P}'$, $\mathcal{Q} \xrightarrow{t_2}_{\Delta} \mathcal{Q}'$ *and* $\mathrm{tn}(\mathcal{P}) \cap \mathrm{tn}(\mathcal{Q}) \subseteq \Delta$. *Then for all* $t \in t_1 \# t_2$ *there exists* $\mathcal{R}$ *such that* $\mathcal{P} \mid \mathcal{Q} \xrightarrow{t}_{\Delta} \mathcal{R} \leftrightsquigarrow (\mathcal{P}' \mid \mathcal{Q}')$.

Prop. 5 is an important but non-trivial result, which requires a proof that transaction structure does not limit communication. For example, let

$$\llbracket a.\mathtt{co}\ k \rhd_k \mathbf{0} \rrbracket \xrightarrow{a}_{\{k,l\}} \mathbf{0} \qquad \text{and} \qquad \llbracket \overline{a}.\mathtt{co}\ k \rhd_l \mathbf{0} \rrbracket \xrightarrow{\overline{a}}_{\{k,l\}} \mathbf{0}$$

Then the parallel composition has the trace

$$\begin{array}{rll}
 & \llbracket a.\mathtt{co}\ k \rhd_k \mathbf{0} \rrbracket & \mid \llbracket \overline{a}.\mathtt{co}\ k \rhd_l \mathbf{0} \rrbracket \\
\xrightarrow{\mathtt{emb}\ k} & \llbracket a.\mathtt{co}\ k \rhd_k \mathbf{0} \rrbracket & \mid \llbracket \llbracket \overline{a}.\mathtt{co}\ k \rhd_l \mathbf{0} \rrbracket \rhd_k \llbracket \overline{a}.\mathtt{co}\ k \rhd_l \mathbf{0} \rrbracket \rrbracket^{\circ} \\
\xrightarrow{\mathtt{emb}\ l} & \llbracket \llbracket a.\mathtt{co}\ k \rhd_l a.\mathtt{co}\ k \rrbracket^{\circ} \rhd_k \mathbf{0} \rrbracket & \mid \llbracket \llbracket \overline{a}.\mathtt{co}\ k \rhd_l \mathbf{0} \rrbracket \rhd_k \llbracket \overline{a}.\mathtt{co}\ k \rhd_l \mathbf{0} \rrbracket \rrbracket^{\circ} \\
\xrightarrow{k(l(\tau)),\mathtt{co}\ l,\mathtt{co}\ k} & \mathbf{0} & \mid \mathbf{0}
\end{array}$$

Hence we can derive the clean trace $\llbracket a.\mathtt{co}\ k \rhd_k \mathbf{0} \rrbracket \mid \llbracket \overline{a}.\mathtt{co}\ k \rhd_l \mathbf{0} \rrbracket \xrightarrow{\tau}_{\{k,l\}} \mathbf{0}$.

Proposition 6 (Completeness w.r.t. ($\hat{\underset{\sim}{\sqsubset}}_{\mathbf{may}}$)). *If* $P \mathrel{\hat{\underset{\sim}{\sqsubset}}_{\text{may}}} Q$ *then* $P \mathrel{\underset{\sim}{\sqsubset}_{\text{tr}}} Q$.

Proof. Standard, using Prop. 5 and an easy unzipping lemma.

Given Prop(s). 4 and 5 and Thm. 1, soundness (Thm. 2) can be proven in a standard way. Completeness (Thm. 3) and conservativity (Thm. 4) follow from Prop. 6 and soundness.

6 Examples

The soundness theorem means that we can prove may-testing equivalences based on weak clean trace inclusion. In this section we give a few examples.

Proposition 3(5). $\llbracket P \rhd_k Q \rrbracket \mathrel{\underset{\sim}{\sqsubset}_{\text{may}}} \tau.P + \tau.Q$.

Proof (outline). Let $\llbracket P \rhd_k Q \rrbracket \overset{t}{\Rightarrow}_{\text{CL}}$, where $t = \mu_1, \ldots, \mu_n$. That is, $\exists \mathcal{R}, \Delta$ such that $\llbracket P \rhd_k Q \rrbracket \xrightarrow{t}_{\Delta} \mathcal{R}$. Either $k \in \Delta$ or $k \notin \Delta$. If $k \in \Delta$ then t corresponds to a trace $k(\widetilde{l}_1(\mu_1))$, …, $k(\widetilde{l}_2(\mu_2))$, …, $\mathtt{co}\ k$ of actions inside k interspersed with broadcast actions and ending on a commit of k. This means that P will have a trace $\widetilde{l}_1(\mu_1)$, …, $\widetilde{l}_2(\mu_2)$, … which corresponds to the *same* clean trace t. On the other hand, if $k \notin \Delta$ then t is the empty trace, or it must correspond to a trace that starts with an abort of k, followed by a trace s of Q. Since the abort is not part of the clean trace, s corresponds to the same clean trace t. Hence $\tau.P + \tau.Q$ includes the weak clean traces of $\llbracket P \rhd_k Q \rrbracket$. □

Example 8. $\llbracket a.b.\mathtt{co}\ k \rhd_k c \rrbracket \mathrel{\underset{\sim}{\sqsubset}_{\text{may}}} a.b + c$ but not vice versa.

Proof. The inclusion follows from 3(5). We can also prove it directly, because as we saw in Sect. 5.1, the set of clean traces of the former process is $\{\epsilon, a, ab, c\}$ while the set of clean traces of the latter is $\{\epsilon, ab, c\}$. □

Theorem 5 (Compositional reasoning). *If $P \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{tr}} Q$ then $P \mid R \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{tr}} Q \mid R$.*

Proof. Since may-testing supports compositional reasoning, this theorem follows directly from soundness and completeness. □

Example 9. Consider the following alternative implementation of the bank and merchant example from the introduction, in which the merchant M' tries to complete the order twice before reporting an error back to the client.

$$\mathit{req}.\,\llbracket \tau.\overline{\mathit{tr}}.(\mathsf{co}\ k \mid \overline{\mathit{ack}}) \rhd_k \llbracket \overline{\mathit{tr}}.(\mathsf{co}\ k \mid \overline{\mathit{ack}}) \rhd_k \overline{\mathit{err}} \rrbracket \rrbracket \mid \mu X.\,\llbracket \mathit{tr}.\tau.\mathsf{co}\ l \rhd_l X \rrbracket$$

By compositional reasoning, we only need to prove the two implementations of the merchant equivalent ($M \eqsim_{\mathrm{may}} M'$) to prove the two systems equivalent.

Intuitively, an observer cannot distinguish between M and M' because when either merchant aborts, the observer is also rolled back: aborts are not detectable. Moreover, the observable behaviour of M', before or after the first abort, equals the behaviour of M.

Formally, the set of weak clean traces of both implementations equals the set $\{\epsilon, \mathit{req}, \mathit{req}\,\overline{\mathit{tr}}, \mathit{req}\,\overline{\mathit{tr}}\,\overline{\mathit{ack}}, \mathit{req}\,\overline{\mathit{err}}\}$. □

7 Programmable Aborts

In TransCCS aborts are entirely non-deterministic. We now turn our attention to *programmable aborts*; i.e. aborts that are triggered by the process through a new language primitive $\mathsf{ab}\ k$. The semantics of this new construct is given by rule R-Prog-Ab, below, replacing rule R-Ab:

$$\textsc{R-Prog-Ab}\qquad \frac{}{\llbracket \mathsf{ab}\ k \mid P \rhd_k Q \rrbracket \to Q}$$

This new language, called TransCCS$^{\mathsf{ab}}$, does not however preserve consistency after an abort. Programmable aborts introduce an undesirable causal dependency between the alternative behaviour of a transaction which follows the abort, and the actions that led to that abort. For example, after the reduction

$$a \mid \llbracket \overline{a}.\mathsf{ab}\ k \rhd_k b.\omega \rrbracket \xrightarrow{\textsc{R-Emb}}\xrightarrow{\textsc{R-Comm}}\xrightarrow{\textsc{R-Prog-Ab}} a \mid \omega$$

a communication on channel b is available *because* a communication on channel a led to an abort; *but* this communication on a is undone. Hence, from the point of view of the left process a the communication has not yet happened, but from the point of view of the transaction it has and led to an abort.

This is more than just a philosophical objection: using transactions such as the above as tests we can show that the basic equivalences in Prop. 1 are not preserved in TransCCS$^{\mathsf{ab}}$. For example, take the two transactions

$$P = \llbracket b.\mathsf{co}\ l \rhd_l \mathbb{0} \rrbracket \qquad Q = \llbracket b.\mathsf{co}\ l + a \rhd_l \mathbb{0} \rrbracket$$

and the same test $T = [\![\overline{a}.\mathtt{ab}\ k \rhd_k b.\omega]\!]$. Then P fails the test T but Q does not, and hence $P \not\eqsim_{\text{may}} Q$, *even though* the a action is never committed:

$$\begin{aligned}
&[\![b.\mathtt{co}\ l + a \rhd_l \mathbb{0}]\!] \mid [\![\overline{a}.\mathtt{ab}\ k \rhd_k b.\omega]\!] \\
\xrightarrow{\text{R-Emb}}\ &[\![(b.\mathtt{co}\ l + a) \mid [\![\overline{a}.\mathtt{ab}\ k \rhd_k b.\omega]\!] \rhd_l T]\!] \\
\xrightarrow{\text{R-Emb}}\ &[\![[\![(b.\mathtt{co}\ l + a) \mid \overline{a}.\mathtt{ab}\ k \rhd_k (b.\mathtt{co}\ l + a) \mid b.\omega]\!] \rhd_l T]\!] \\
\xrightarrow{\text{R-Comm}}\ &[\![[\![\mathtt{ab}\ k \rhd_k (b.\mathtt{co}\ l + a) \mid b.\omega]\!] \rhd_l T]\!] \\
\xrightarrow{\text{R-Prog-Ab}}\ &[\![(b.\mathtt{co}\ l + a) \mid b.\omega \rhd_l T]\!] \xrightarrow{\text{R-Comm}} [\![\mathtt{co}\ l \mid \omega \rhd_l T]\!] \xrightarrow{\text{R-Co}} \omega
\end{aligned}$$

We can recover the preservation of consistency, however, by restricting all transactions to be restarting, i.e. of the form $\mu X.\,[\![P \rhd_k X]\!]$; we call this language TransCCS$^{\mu\mathtt{ab}}$. Unfortunately, it is difficult to reason directly about TransCCS$^{\mu\mathtt{ab}}$ processes, since the language is not closed under reduction: after a restarting transaction unfolds and its default process reduces, we are left with a TransCCS$^{\mathtt{ab}}$ transaction whose default and alternative processes are different. However, we can give a theory for TransCCS$^{\mu\mathtt{ab}}$ through a *fully-abstract translation* to TransCCS, and reason about the behaviour of TransCCS$^{\mu\mathtt{ab}}$ processes via the translation.

We consider the translation $\{\!|\cdot|\!\}$: TransCCS$^{\mu\mathtt{ab}}$ → TransCCS, which maps $\mathtt{ab}\ k$ to $\mathbb{0}$ and is the identity on all other constructs. We use the annotation "$\mu\mathtt{ab}$" to refer to the semantics of TransCCS$^{\mu\mathtt{ab}}$. The important property of the translation is that it preserves barbs. From that and the results of Sect. 5 we derive the theorems of soundness and full abstraction.

Proposition 7. *$P{\Downarrow}^{\mu\mathtt{ab}}_{\omega}$ iff $\{\!|P|\!\}{\Downarrow}_{\omega}$.*

Theorem 6 (Full Abstraction of $\{\!|\cdot|\!\}$). *$P \sqsubseteq^{\mu\mathtt{ab}}_{\text{may}} Q$ iff $\{\!|P|\!\} \sqsubseteq_{\text{may}} \{\!|Q|\!\}$.*

8 Related Work

To the extent of our knowledge there is little related work on modelling automatic error recovery of communicating systems. Most work has either focused on models for isolated transactions [4,17], including software transactional memory [18,1], or compensation-based transactions [5,9,8,20,11] where error recovery must be programmed explicitly.

TransCCS is motivated by the long literature on implementing distributed systems with automatic error recovery (e.g. [16,13,22,23]) and their verification in process calculi (such as [3,6,21]). This work, however, is only indirectly related to ours as we are not proposing a mechanism for *implementing* automatic rollback recovery but rather a way to give high-level specifications of, and reason about, distributed systems that rely on automatic error recovery.

The only other language that we are aware of in which non-isolated transactions can be modelled is Reversible CCS [14]. RCCS extends CCS with the notion of *reversible actions* (written $a, \overline{a}, \ldots$) and *irreversible actions* (written $\underline{a}, \underline{\overline{a}}, \ldots$) past which processes cannot be rolled back. This can be used to model simple (non-nested) transactions; for example, a transaction superficially similar to $\mu X.\, [\![a + b.\mathsf{co}\ k \rhd_k X]\!]$ can be written as $(a + \underline{b})$ in RCCS. The dynamic behaviour of these two terms is significantly different however: the RCCS transaction is not in charge of when it commits. An irreversible action $\underline{\overline{a}}$ by an observer can interact with the reversible action a of the transaction, forcing the transaction to commit. Thus the test $\underline{\overline{a}}.\underline{\omega}$ succeeds when paired with the above transaction in RCCS, but must fail in TransCCS. The same observer can distinguish $\mathbb{0}$ from the transaction $[\![a \rhd_k \mathbb{0}]\!]$ (in our syntax), which are indistinguishable in TransCCS. Hence a testing theory of RCCS would have to take into account the non-committing traces of transactions, in contrast to our theory for TransCCS in which the behaviour of a transaction only depends on its committing behaviour.

9 Conclusions

We presented a novel language construct called communicating transactions, which makes it possible to describe the behaviour of distributed systems with automatic error recovery at a high level of abstraction. We believe that support for communicating transactions may be beneficial in the design and implementation of complex distributed systems such as web services.

We introduced TransCCS, an extension of CCS with this construct. To the extent of our knowledge TransCCS is the first calculus which encapsulates both rollback recovery and coordinated checkpointing. We gave simple semantics to TransCCS and developed a basic behavioural theory, based on non prefix-closed sets of traces, that characterizes may-testing. We used the theory to prove a number of interesting laws and reason compositionally about example systems. We also studied TransCCS$^{\mu\mathsf{ab}}$, a variant of the language with programmable aborts, and gave a fully-abstract translation to TransCCS.

We plan to study the must-testing or fair-testing [25] theory of TransCCS in order to be able to specify liveness properties in the presence of aborts; we expect that the translation from TransCCS$^{\mu\mathsf{ab}}$ into TransCCS from Sect. 7 will not be fully abstract with respect to these testing preorders. We also plan to extend our work to the π-calculus and other behavioural equivalences such as bisimulation. Finally, we intent to investigate the usefulness of the construct of communicating transactions in a more realistic programming language.

References

1. Acciai, L., Boreale, M., Zilio, S.D.: A concurrent calculus with atomic transactions. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 48–63. Springer, Heidelberg (2007)
2. Benslimane, D., Dustdar, S., Sheth, A.: Services mashups: The new generation of web applications. IEEE Internet Computing 12, 13–15 (2008)
3. Berger, M., Honda, K.: The two-phase commitment protocol in an extended π-calculus. In: EXPRESS. ENTCS, vol. 39, pp. 21–46. Elsevier, Amsterdam (2003)
4. Black, A.P., Cremet, V., Guerraoui, R., Odersky, M.: An equational theory for transactions. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 38–49. Springer, Heidelberg (2003)

5. Bocchi, L.: Compositional nested long running transactions. In: Wermelinger, M., Margaria-Steffen, T. (eds.) FASE 2004. LNCS, vol. 2984, pp. 194–208. Springer, Heidelberg (2004)
6. Bocchi, L., Wischik, L.: A process calculus of atomic commit. In: WS-FM. ENTCS, vol. 105, pp. 119–132. Elsevier Science Publishers, Amsterdam (2004)
7. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web services architecture, W3C Working Group Note (February 2004)
8. Bruni, R., Melgratti, H., Montanari, U.: Theoretical foundations for compensations in flow composition languages. In: POPL, pp. 209–220. ACM, New York (2005)
9. Bruni, R., Melgratti, H., Montanari, U.: Nested commits for mobile calculi: extending Join. In: IFIP-TCS, pp. 569–582. Kluwer Academic Publishers, Dordrecht (2004)
10. Cabrera, L.F., et al.: Web services business activity framework (WS-BusinessActivity). Whitepaper (August 2005)
11. Caires, L., Ferreira, C., Vieira, H.T.: A process calculus analysis of compensations. In: Kaklamanis, C., Nielson, F. (eds.) TGC 2008. LNCS, vol. 5474, pp. 87–103. Springer, Heidelberg (2009)
12. Ceponkus, A., Dalal, S., Fletcher, T., Furniss, P., Green, A., Pope, B.: Business transaction protocol. In: OASIS Committee Specification (June 2002)
13. Chandy, K.M., Lamport, L.: Distributed snapshots: determining global states of distributed systems. ACM Trans. Comp. Syst. 3(1), 63–75 (1985)
14. Danos, V., Krivine, J.: Transactions in RCCS. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 398–412. Springer, Heidelberg (2005)
15. De Nicola, R., Hennessy, M.C.B.: Testing equivalences for processes. Theoretical Computer Science 34(1-2), 83–133 (1984)
16. Elnozahy, E.N.M., Alvisi, L., Wang, Y.M., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. ACM Comp. Surv. 34(3), 375–408 (2002)
17. Gorrieri, R., Marchetti, S., Montanari, U.: A^{2}CCS: atomic actions for CCS. Theor. Comp. Sci. 72(2-3), 203–223 (1990)
18. Harris, T., Marlow, S., Peyton-Jones, S., Herlihy, M.: Composable memory transactions. In: PPoPP, pp. 48–60. ACM, New York (2005)
19. Little, M.: Transactions and web services. Commun. ACM 46(10), 49–54 (2003)
20. Lucchi, R., Mazzara, M.: A pi-calculus based semantics for WS-BPEL. Journal of Logic and Algebraic Programming 70(1), 96–118 (2007)
21. Nestmann, U., Fuzzati, R., Merro, M.: Modeling consensus in a process calculus. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 399–414. Springer, Heidelberg (2003)
22. Nett, E., Mock, M.: How to commit concurrent, non-isolated computations. In: FTDCS, pp. 343–353. IEEE Comp. Soc., Los Alamitos (1995)
23. Park, T., Lee, I., Yeom, H.Y.: An efficient causal logging scheme for recoverable distributed shared memory systems. Parallel Computing 28(11), 1549–1572 (2002)
24. Rathke, J., Sassone, V., Sobocinski, P.: Semantic barbs and biorthogonality. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 302–316. Springer, Heidelberg (2007)
25. Rensink, A., Vogler, W.: Fair testing. Inf. and Comp. 205(2), 125–198 (2007)
26. de Vries, E., Koutavas, V., Hennessy, M.: Communicating transactions—technical appendix (April 2010), http://www.scss.tcd.ie/Edsko.de.Vries
27. Weikum, G., Vossen, G.: Transactional information systems. In: Distributed Transaction Recovery, ch. 20. Morgan Kaufmann Publishers Inc., San Francisco (2001)

Consistent Correlations for Parameterised Boolean Equation Systems with Applications in Correctness Proofs for Manipulations

Tim A.C. Willemse

Department of Mathematics and Computer Science,
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract. We introduce the concept of *consistent correlations* for parameterised Boolean equation systems (PBESs), motivated largely by the laborious proofs of correctness required for most manipulations in this setting. Consistent correlations focus on relating the equations that occur in PBESs, rather than their solutions. For a fragment of PBESs, consistent correlations are shown to coincide with a recently introduced form of bisimulation. Finally, we show that bisimilarity on *processes* induces consistent correlations on PBESs encoding model checking problems. We apply our theory to two example manipulations from the literature.

1 Introduction

First introduced in [12] as a first-order extension of Boolean equation systems [11], *parameterised Boolean equation systems* (henceforth referred to as *equation systems*), provide a uniform view on a variety of verification problems for reactive systems. Problems as diverse as equivalence checking problems for a variety of process equivalences [1], model checking problems for symbolic transition systems [8] and real-time systems [17], and static analysis of code [3] have been encoded in this framework. In all cases, the solution to the encoded problem is reflected in the solution to the equation system. Advanced toolsets, e.g., CADP [4] and mCRL2 [7], have been built around equation system technology.

Various techniques for solving equation systems have been developed, such as the combination of *Gauß elimination* and *symbolic approximation* [8], *instantiation* [2,13], and solving by means of *pattern matching* [9]. The undecidability of solving equation systems in general, however, implies that there is no universally applicable method to solve all encoded verification problems.

The undecidability of the problem has prompted research to shift to finding manipulations that simplify the task of solving equation systems. In fact, one of the merits of the framework of equation systems is that it facilitates the development of such techniques. Noteworthy manipulations are the static analysis techniques discussed in [14], which add to the efficacy of instantiation; the use of global invariants [15] to ensure the termination of symbolic approximation; and the use of minimisation techniques [10], accelerating the solving of Boolean

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 584–598, 2010.

equation systems. Unfortunately, devising new manipulations has proven to be a laborious process, undermining steady progress in this area.

The bottleneck in developing a new manipulation is more often than not its correctness proof. As already noted in [9], such proofs tend to require a good deal of ingenuity, meticulous bookkeeping, inductions on the length of equation systems and transfinite approximations of the involved fixed point expressions. We propose a new equivalence on equation systems, called a *consistent correlation*, which enables one to lift the reasoning in correctness proofs for manipulations to a more natural level. Apart from facilitating such correctness proofs –providing a likely boost to the development of novel manipulations– consistent correlations contribute to our understanding of the notion of solution to equation systems.

Our contributions are as follows. We first define the notion of a consistent correlation and explain its rationale. Next, we present our main result, showing that each consistent correlation refines the solution to an equation system, relating only predicate variable instances that receive the same truth value in the solution. We continue by proving that consistent correlations coincide with *idempotence-identifying bisimulations* [10], a variation on strong bisimulation on dependency graphs underlying *closed* Boolean equation systems in *standard recursive form*. Note that this fragment of equation systems is interesting in its own right, as it corresponds one-to-one with Parity Games [5], a game-based formalisms often used for studying verification problems. The relation between idempotence-identifying bisimilarity and consistent correlations also serves as the basis for our decidability results for consistent correlations on *open* Boolean equation systems in *standard form*. Finally, we show that strong bisimilarity on processes induces consistent correlations in equation systems that encode model checking problems, following the encoding of [8]. This is a good indication that the concept of a consistent correspondence is not overly discriminating. We apply our theory to two manipulations from the literature, illustrating that a natural meta-argument can be used to replace the original extensive correctness proofs.

Outline. Section 2 gives a brief introduction into parameterised Boolean equation systems. Consistent correlations are introduced in Section 3. Section 4 relates consistent correlations to idempotence-identifying bisimulations, and in Section 5, we establish the link between bisimilarity on processes and consistent correlations in equation systems. In Section 6, we illustrate the theory through two manipulations from the literature. We finish with an outlook on future work.

2 Preliminaries

We give a brief overview of the relevant notations and concepts for understanding the results described in this paper; for a detailed exposition, we refer to [9].

2.1 Data

We work in the setting of *abstract data types*, i.e., we assume that there are nonempty data sorts and operations on these sorts. We typically use letters

$D_1, D_2, \ldots$ to denote data sorts. Furthermore, we assume to have a set $\mathcal{D}$ of *sorted data variables*, with typical elements $d, d_1, \ldots$, *etc.* With every sort D we associate a semantic set $\mathbb{D}$ such that every term of sort D and all the operations on the sort D can be mapped to the elements and operations of $\mathbb{D}$ they represent.

We assume the existence of an interpretation function $[\![\_]\!]$ that maps every closed data term t of sort D to the data element $[\![t]\!]$ of $\mathbb{D}$ it represents. For open terms we use an *environment* ε that maps each variable from $\mathcal{D}$ to a data element of the associated type. The interpretation $[\![t]\!]\,\varepsilon$ of an open term t is given by $\varepsilon(t)$, where ε is extended to terms in the standard way.

For convenience, we assume the existence of a sort $B = \{\top, \bot\}$ representing the Booleans $\mathbb{B}$, and a sort $N = \{0, 1, 2, \ldots\}$ representing the natural numbers $\mathbb{N}$. For these sorts, we assume the usual operators are available. We do not write constants and operators in the syntactic domain any different from their semantic counterparts. For example, we have $\mathbb{B} = \{\top, \bot\}$ and the syntactic operator $\_\wedge\_{:}B \times B \to B$ corresponds to the usual, semantic conjunction $\_\wedge\_{:}\mathbb{B} \times \mathbb{B} \to \mathbb{B}$.

2.2 Parameterised Boolean Equation Systems

Equation systems are finite sequences of fixed point equations ranging over predicate formulae. The latter are formalised below.

Definition 1. *The set of* predicate formulae *is defined by the grammar below:*

$$\phi, \psi ::= b \mid X(e) \mid \phi \wedge \psi \mid \phi \vee \psi \mid \forall d{:}D.\ \phi \mid \exists d{:}D.\ \phi$$

here, b is a data term of sort B, X is a predicate variable of sort $D_X \to B$ taken from some sufficiently large set $\mathcal{P}$ of predicate variables, d is a data variable of sort D and e is some data term of sort D_X.

Predicate variables of sort B are referred to as *proposition* variables. *Proposition formulae* are predicate formulae containing only conjunctions, disjunctions, Boolean constants and proposition variables.

The *signature* of a predicate variable X of sort $D_X \to B$, denoted $\mathsf{sig}(X)$, is the product $\{X\} \times \mathbb{D}_X$. We lift the notion of a signature of a predicate variable to sets of variables $P \subseteq \mathcal{P}$ in the natural way:

$$\mathsf{sig}(P) \mathrel{\hat{=}} \bigcup_{X \in P} \mathsf{sig}(X)$$

By convention, we write $v_X \in \mathsf{sig}(X)$ instead of $(X, v) \in \mathsf{sig}(X)$, emphasising the importance of data values over predicate variable names; for proposition variables X, we write $X \in \mathsf{sig}(X)$.

Definition 2. *The interpretation of ϕ in the context of environments θ for predicate variables and ε for data variables is denoted $[\![\phi]\!]\,\theta\varepsilon$, where:*

$$\begin{array}{llll}
[\![b]\!]\,\theta\varepsilon & \hat{=}\ \varepsilon(b) & [\![X(e)]\!]\,\theta\varepsilon & \hat{=}\ \theta(X)(\varepsilon(e)) \\
[\![\phi \wedge \psi]\!]\,\theta\varepsilon & \hat{=}\ [\![\phi]\!]\,\theta\varepsilon \wedge [\![\psi]\!]\,\theta\varepsilon & [\![\phi \vee \psi]\!]\,\theta\varepsilon & \hat{=}\ [\![\phi]\!]\,\theta\varepsilon \vee [\![\psi]\!]\,\theta\varepsilon \\
[\![\forall d{:}D.\ \phi]\!]\,\theta\varepsilon & \hat{=}\ \forall v \in \mathbb{D}.\ [\![\phi]\!]\,\theta\varepsilon[v/d] & [\![\exists d{:}D.\ \phi]\!]\,\theta\varepsilon & \hat{=}\ \exists v \in \mathbb{D}.\ [\![\phi]\!]\,\theta\varepsilon[v/d]
\end{array}$$

Note that we write $\varepsilon[v/d]$ to represent the environment ε' which is defined as $\varepsilon'(d') \mathrel{\hat{=}} \varepsilon(d')$ for $d' \neq d$ and $\varepsilon'(d) \mathrel{\hat{=}} v$.

Observe that our formulae are given directly in *positive form*, i.e., negation only occurs at the level of Boolean expressions. This is to ensure that the interpretations are monotone in the valuation of the predicate variables.

Definition 3. *Equation systems are defined by the following grammar:*

$$\mathcal{E} ::= \epsilon \mid \big(\nu X(d_X{:}D_X) = \phi_X\big)\ \mathcal{E} \mid \big(\mu X(d_X{:}D_X) = \phi_X\big)\ \mathcal{E}$$

The constant ϵ denotes the empty equation system; μ and ν are the least fixed point and greatest fixed point signs, respectively; X is a sorted predicate variable of sort $D_X \rightarrow B$, d_X is a formal parameter, and ϕ_X is a predicate formula.

Henceforth, we write ϕ_X for the predicate formula occurring at the right-hand side in the equation for X, d_X for the formal data parameters occurring at the left-hand side of X's equation and D_X for the sort of these parameters. An equation system $\mathcal{E}$ is said to be a *Boolean equation system* if all right-hand sides of the equations in $\mathcal{E}$ are proposition formulae. Boolean equation systems form an interesting, decidable fragment that have been studied extensively, see e.g. [11]; they are closely related to Parity Games [5].

Let $\mathsf{occ}(\phi)$ denote the set of predicate variables that occur in a predicate formula ϕ. Extended to equation systems, $\mathsf{occ}(\mathcal{E})$ is the union of all variables occurring at the right-hand side of equations in $\mathcal{E}$. The set of *binding predicate variables* of $\mathcal{E}$, denoted $\mathsf{bnd}(\mathcal{E})$, is the set of predicate variables occurring at the left-hand side of $\mathcal{E}$'s equations. An equation system $\mathcal{E}$ is said to be *well-formed* iff every binding predicate variable occurs at the left-hand side of precisely one equation of $\mathcal{E}$. We only consider well-formed equation systems.

A *block* is a non-empty equation system $\mathcal{B}$ consisting entirely of like-signed fixed point equations. Given an equation system $\mathcal{E}$, a block $\mathcal{B}$ occurring in $\mathcal{E}$ is *maximal* if its neighbouring equations in $\mathcal{E}$ are of a different sign than the equations in block $\mathcal{B}$. We denote the i-th maximal block of $\mathcal{E}$ by $\mathcal{E}\rceil i$.

Definition 4. *Let $\mathcal{E}$ be an equation system. We define a total ordering $\trianglelefteq_{\mathcal{E}} \subseteq \mathsf{bnd}(\mathcal{E}) \times \mathsf{bnd}(\mathcal{E})$ by setting $X \trianglelefteq_{\mathcal{E}} X'$ iff $i \leq j$ for $X \in \mathsf{bnd}(\mathcal{E}\rceil i), X' \in \mathsf{bnd}(\mathcal{E}\rceil j)$.*

Intuitively, $X \trianglelefteq_{\mathcal{E}} X'$ orders the predicate variables according to the maximal block in which they occur, abstracting from the internal ordering in blocks.

Let $\lambda d_X.\phi_X$ abbreviate the syntactic functional $\lambda d_X{:}D_X.\phi_X$, i.e., we drop the sort information. The interpretation of $\lambda d_X.\phi_X$, denoted $[\![\lambda d_X.\phi_X]\!]\theta\varepsilon$, is given by the following functional:

$$(\lambda v{\in}\mathbb{D}_X.\ [\![\phi_X]\!]\theta\varepsilon[v/d_X])$$

Let $\mathbb{B}^{\mathbb{D}_X}$ denote the set of functions $f{:}\mathbb{D}_X \rightarrow \mathbb{B}$. We define an ordering $\sqsubseteq$ on functions $f, g \in \mathbb{B}^{\mathbb{D}_X}$ as follows: $f \sqsubseteq g$ iff for all $v \in \mathbb{D}_X$, $f(v)$ implies $g(v)$. Clearly, the set $(\mathbb{B}^{\mathbb{D}_X}, \sqsubseteq)$ is a complete lattice. The predicate transformer associated to the functional $[\![\lambda d_X.\phi_X]\!]\theta\varepsilon$ is the monotone operator given by:

$$(\lambda f{\in}\mathbb{B}^{\mathbb{D}_X}.\ [\![\lambda d_X.\phi_X]\!]\theta[f/X]\varepsilon)$$

Let σ denote an arbitrary fixed point sign. The extremal fixed point of the above predicate transformer is denoted $\sigma f {\in} \mathbb{B}^{\mathbb{D}_X}.\ [\![\lambda d_X.\phi_X]\!]\,\theta[f/X]\varepsilon$.

Definition 5. *The* solution *to an equation system, given a context of predicate and data environments θ, ε, is defined as follows:*

$$\begin{aligned} [\![\epsilon]\!]\,\theta\varepsilon &\mathrel{\hat{=}} \theta \\ [\![(\sigma X(d_X{:}D_X) = \phi_X)\ \mathcal{E}]\!]\,\theta\varepsilon &\mathrel{\hat{=}} [\![\mathcal{E}]\!]\Big(\theta\big[\sigma f{\in}\mathbb{B}^{\mathbb{D}_X}.\ [\![\lambda d_X.\phi_X]\!](\,[\![\mathcal{E}]\!]\,\theta[f/X]\varepsilon)\varepsilon/X\big]\Big)\varepsilon \end{aligned}$$

Informally, the solution to an equation system can be characterised as follows:

- The solution to an equation system is an assignment to the binding predicate variables that indeed verifies each equation semantically, i.e.:

  $$[\![\mathcal{E}]\!]\,\theta\varepsilon(X) = [\![\lambda d_X.\phi_X]\!](\,[\![\mathcal{E}]\!]\,\theta\varepsilon)\varepsilon$$

  In particular, $[\![\mathcal{E}]\!]\,\theta\varepsilon \in \Theta_{ful}$, where Θ_{ful} is the set of all *fulfilments*:

  $$\Theta_{ful} \mathrel{\hat{=}} \{\theta \mid \exists \varepsilon : \forall X \in \mathsf{bnd}(\mathcal{E}) :\ \theta(X) = [\![\lambda d_X.\phi_X]\!]\,\theta\varepsilon\}$$

- The fixed point signs of equations that are left-most take priority over the fixed point signs of the equations that follow; this ensures unicity (see e.g. [11]). For instance, $(\mu X = Y)(\nu Y = X)$ has solution $X = \bot, Y = \bot$.

Equation systems $\mathcal{E}$ that satisfy $\mathsf{occ}(\mathcal{E}) \subseteq \mathsf{bnd}(\mathcal{E})$ and in which none of the functionals $\lambda d_X.\phi_X$ contain free data variables, are said to be *closed.* For closed $\mathcal{E}$, we have for all $X \in \mathsf{bnd}(\mathcal{E})$ and all $\theta, \theta', \varepsilon$, $[\![\mathcal{E}]\!]\,\theta\varepsilon(X) = [\![\mathcal{E}]\!]\,\theta'\varepsilon'(X)$; by convention, we therefore write $[\![\mathcal{E}]\!]$ rather than $[\![\mathcal{E}]\!]\,\theta\varepsilon$ for such equation systems. Note that most practical verification problems result in closed equation systems.

3 Consistent Correlations

One of the merits of using equation systems for verification is the potential to manipulate the equations symbolically. The correctness of a manipulation often requires showing that the solution to an equation system *after* some manipulation m relates to the solution prior to the manipulation, i.e.:

$$[\![\mathcal{E}]\!]\,\theta\varepsilon(X)(v) = [\![\mathsf{m}(\mathcal{E})]\!]\,\theta\varepsilon(\mathsf{m}(X))(\mathsf{m}(v))$$

Most correctness proofs, see e.g. [9,2,15,14], resort to the inductive definition of the solution to an equation system at some point (indeed, the very proof required for the main theorem in this section is a point in case). This is not only very laborious, it also obfuscates the structure of the proof. In this section, we introduce *consistent correlations*, allowing one to raise the level of abstraction in such proofs.

Avoiding the inevitable notational overhead, we first introduce consistent correlations on single equation systems. We subsequently proceed to show how they can be used to relate different equation systems.

Definition 6. *Let $R \subseteq \mathsf{sig}(\mathcal{P}) \times \mathsf{sig}(\mathcal{P})$ be an arbitrary relation. An environment θ is an R-*correlation *iff $v_X R v'_{X'}$ implies $\theta(X)(v) = \theta(X')(v')$.*

Let Θ_R, for given relation R, denote the set of R-correlating environments. Proving that two (possibly different) equations in a single system $\mathcal{E}$ reflect one another's solution now can be phrased as the problem of proving whether for a desired relation R, the solution $\llbracket \mathcal{E} \rrbracket \theta\varepsilon$ is an R-correlation: for $X, X' \in \mathsf{bnd}(\mathcal{E})$, $\llbracket \mathcal{E} \rrbracket \theta\varepsilon(X)(v) = \llbracket \mathcal{E} \rrbracket \theta\varepsilon(X')(v')$ follows from the existence of some relation R such that $v_X R v'_{X'}$ and $\llbracket \mathcal{E} \rrbracket \theta\varepsilon \in \Theta_R$. We next postulate restrictions on the relation R that guarantee that the latter holds.

Definition 7. *Let $\mathcal{E}$ be an equation system. A relation $R \subseteq \mathsf{sig}(\mathcal{P}) \times \mathsf{sig}(\mathcal{P})$ is a* consistent correlation *on $\mathcal{E}$, if for $X, X' \in \mathsf{bnd}(\mathcal{E})$, $v_X R v'_{X'}$ implies:*

1. *for all i, $X \in \mathsf{bnd}(\mathcal{E} \rceil i)$ iff $X' \in \mathsf{bnd}(\mathcal{E} \rceil i)$;*
2. *for all $\theta \in \Theta_R$, ε, we have $\llbracket \phi_X \rrbracket \theta\varepsilon[v/d_X] = \llbracket \phi_{X'} \rrbracket \theta\varepsilon[v'/d_{X'}]$*

For $X, X' \in \mathsf{bnd}(\mathcal{E})$, we say v_X and $v'_{X'}$ consistently correlate*, notation $v_X \doteqdot v'_{X'}$ iff $v_X R v'_{X'}$ for some consistent correlation $R \subseteq \mathsf{sig}(\mathsf{bnd}(\mathcal{E})) \times \mathsf{sig}(\mathsf{bnd}(\mathcal{E}))$.*

Intuitively, an R-correlating environment θ asserts that R-related pairs have the same potential to be *true* or *false*. A consistent correlation R, additionally asserts that this allows one to prove that in a given equation systems, also the right-hand sides of related pairs have the same potential to be *true* or *false*; i.e., also the right-hand sides of correlating pairs correlate.

Observe that the second requirement in the characterisation of a consistent correlation does not imply that a consistent correlation R on $\mathcal{E}$ induces only R-correlating environments θ that are fulfilments of the equations in $\mathcal{E}$: we do not have $\Theta_R \subseteq \Theta_{ful}$. Still, as we shall prove in this section, the solution to an equation system is in the intersection of a consistent correlation and the set of all fulfilments, as visualised by Figure 1.

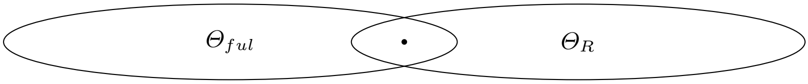


Fig. 1. The set of R-correlating environments in relation to the environments verifying each equation, with the solution of an equation system in the intersection of all

The relation $\doteqdot$ is an equivalence relation on the signature of the binding predicate variables in an equation system. It is easy to check that arbitrary unions of consistent correlations on $\mathcal{E}$ are again consistent correlations on $\mathcal{E}$. The relation $\doteqdot$ can be phrased as the greatest fixed point of the following functional:

$$\begin{aligned} \mathcal{F}(R) \mathrel{\hat{=}} \big\{ (v_X, v'_{X'}) \in \mathsf{sig}(\mathsf{bnd}(\mathcal{E})) \times \mathsf{sig}(\mathsf{bnd}(\mathcal{E})) \mid \\ \exists i : X, X' \in \mathsf{bnd}(\mathcal{E} \rceil i) \\ \text{and } \forall \theta \in \Theta_R, \varepsilon : \ \llbracket \phi_X \rrbracket \theta\varepsilon[v/d_X] = \llbracket \phi_{X'} \rrbracket \theta\varepsilon[v'/d_{X'}] \big\} \end{aligned}$$

We next show that, indeed, any consistent correlation R on $\mathcal{E}$ guarantees that the solution to $\mathcal{E}$ is also an R-correlating environment. This confirms that R-related pairs in an equation system have the same solution.

Theorem 1. *Let $\mathcal{E}$ be an equation system. Let $R \subseteq \mathsf{sig}(\mathcal{P}) \times \mathsf{sig}(\mathcal{P})$ be a consistent correlation on $\mathcal{E}$. Then for all $\theta \in \Theta_R$ and all ε, we have $[\![\mathcal{E}]\!]\theta\varepsilon \in \Theta_R$.*

Proof. The proof proceeds by an induction on the number of maximal blocks in $\mathcal{E}$. The inductive step requires a transfinite approximation of the solution to the first maximal block in $\mathcal{E}$. □

The restriction of environments θ to those in Θ_R, in Theorem 1, is not a limitation in practice: for most equation systems, we are interested in relating only their binding variables, i.e., we are interested in establishing whether $v_X \doteqdot v'_{X'}$. We therefore have the following corollary.

Corollary 1. *Let $\mathcal{E}$ be an equation system. Then for all θ, ε, $[\![\mathcal{E}]\!]\theta\varepsilon \in \Theta_{\doteqdot}$.*

As a result of this corollary, we have $[\![\mathcal{E}]\!] \in \Theta_{\doteqdot}$ for closed equation systems $\mathcal{E}$.

The notion of a consistent correlation, and the preceding theorem can easily be lifted to a relation *between* equation systems that are in some sense *compatible*: free predicate variables in one system should not occur bound in the other system, and the set of binding variables of both systems should be disjoint.

Definition 8. *Let $\mathcal{E}, \mathcal{E}'$ be equation systems. Then $\mathcal{E}$ and $\mathcal{E}'$ are* compatible *iff:*

$$\mathsf{bnd}(\mathcal{E}) \cap \mathsf{bnd}(\mathcal{E}') = \mathsf{bnd}(\mathcal{E}) \cap \mathsf{occ}(\mathcal{E}') = \mathsf{bnd}(\mathcal{E}') \cap \mathsf{occ}(\mathcal{E}) = \emptyset$$

The compatibility requirement is rather benign, as in most practical cases, it can be achieved via a suitable renaming.

Definition 9. *Let $\mathcal{E}, \mathcal{E}'$ be compatible equation systems. A relation $R \subseteq \mathsf{sig}(\mathcal{P}) \times \mathsf{sig}(\mathcal{P})$ is a consistent correlation between $\mathcal{E}$ and $\mathcal{E}'$ if R is a consistent correlation on some equation system $\mathcal{F}$ consisting of the equations of $\mathcal{E}$ and $\mathcal{E}'$ satisfying:*

$$Z \in \mathcal{F}\rceil i \textit{ implies } Z \in \mathcal{E}\rceil i \textit{ or } Z \in \mathcal{E}'\rceil i$$

For $X, X' \in \mathsf{bnd}(\mathcal{E}\mathcal{E}')$, v_X and $v'_{X'}$ consistently correlate*, denoted $v_X \doteqdot v'_{X'}$ iff $v_X R v'_{X'}$ for some consistent correlation $R \subseteq \mathsf{sig}(\mathsf{bnd}(\mathcal{E}\mathcal{E}')) \times \mathsf{sig}(\mathsf{bnd}(\mathcal{E}\mathcal{E}'))$ between $\mathcal{E}$ and $\mathcal{E}'$.*

The restriction on $\mathcal{F}$ effectively requires that $\mathcal{E}$ and $\mathcal{E}'$ have the same fixed-point blocks. One can weaken this requirement to $\trianglelefteq_{\mathcal{E}} \cup \trianglelefteq_{\mathcal{E}'} \subseteq \trianglelefteq_{\mathcal{F}}$, which ensures that the block-ordering of $\mathcal{E}$ and $\mathcal{E}'$ is reflected in that of $\mathcal{F}$. Theorem 1 still allows one to conclude that pairs that can be related through a consistent correlation between $\mathcal{E}$ and $\mathcal{E}'$ have the same solution. Unfortunately, the relation $\doteqdot$, as defined above, would no longer be an equivalence between equation systems:

Example 1. Consider the equation systems $\mathcal{E} \equiv (\nu X = X)(\mu Y = Y)(\nu Z = Z)$ and $\mathcal{E}' \equiv (\nu W = W)$. Following Def. 9, we find $X \doteqdot W$, as witnessed by, e.g.,

$\mathcal{F} \equiv (\nu X = X)(\nu W = W)(\mu Y = Y)(\nu Z = Z)$. Requiring $\trianglelefteq_{\mathcal{E}} \cup \trianglelefteq_{\mathcal{E}'} \subseteq \trianglelefteq_{\mathcal{F}}$ would enable us to derive $Z \doteqdot W$, as witnessed by $\mathcal{F} \equiv (\nu X = X)(\mu Y = Y)(\nu Z = Z)(\nu W = W)$. But, clearly, we can never derive $X \doteqdot Z$. Note that, using Theorem 1, one can still prove that the solution to X, Z and W are the same. □

The following theorem lifts Theorem 1 to consistent correlations between equation systems.

Theorem 2. *Let $\mathcal{E}, \mathcal{E}'$ be compatible equation systems. Then for all $X \in \mathsf{bnd}(\mathcal{E})$, $X' \in \mathsf{bnd}(\mathcal{E}')$ and all $\theta \in \Theta_{\doteqdot}$, we have*

$$v_X \doteqdot v'_{X'} \textit{ implies } [\![\mathcal{E}]\!]\theta\varepsilon(X)(v) = [\![\mathcal{E}']\!]\theta\varepsilon(X')(v')$$

Example 2. Consider the closed equations $\nu X_1(n{:}N) = \mathsf{even}(n) \wedge X_1(n+2)$ and $\nu X_2(b{:}B) = b \wedge X_2(b)$. We find that $v_{X_1} \doteqdot \mathsf{even}(v)_{X_2}$ for all $v \in \mathbb{N}$. Following Theorem 2, this implies that for all $v \in \mathbb{N}$:

$$\begin{aligned} & [\![\nu X_1(n{:}N) = \mathsf{even}(n) \wedge X_1(n+2)]\!]\,(X_1)(v) \\ = \; & [\![\nu X_2(b{:}B) = b \wedge X_2(b)]\!]\,(X_2)(\mathsf{even}(v)) \end{aligned}$$

It should be noted that the solution to X_2 can easily be computed via a transformation to a Boolean equation system (see also Section 6). This immediately leads to the solution $X_2(c) = c$, for $c \in \mathbb{B}$. Using the above relation, we thereby instantly provide a solution to X_1 as well. □

4 Consistent Correlations on Boolean Equation Systems

We next establish a correspondence between consistent correlations and a variation of bisimulation for *dependency graphs* that underly Boolean equation systems, see [10]. Such dependency graphs are basically Parity Games [5] stripped from their concept of winning. We first fix some additional terminology.

Definition 10. *Let $\mathcal{E}$ be a Boolean equation system.*

- *$\mathcal{E}$ is in* standard form *if none of its proposition formulae combine $\wedge$ and $\vee$;*
- *$\mathcal{E}$ is in* recursive form *if none of its equations contain $\top$ and $\bot$.*

If a Boolean equation system is both in standard form and in recursive form, we say it is in *standard recursive form*. Each Boolean equation system can be converted to a Boolean equation system in standard recursive form, reflecting the solution to the original Boolean equation system; the transformation is straightforward, and can be found in e.g. [11].

Definition 11. *Let $\mathcal{E}$ be a closed Boolean equation system in standard recursive form. Set $\delta = 0$ if $\mathcal{E}\rceil 1$ is a greatest fixed point block, and $\delta = 1$ otherwise. The* dependency graph *of $\mathcal{E}$ is a tuple $\langle V, \to, r, l\rangle$, where:*

- *$V = \{v_X \mid X \in \mathsf{bnd}(\mathcal{E})\}$ is a set of vertices;*
- *$\to \subseteq V \times V$ is the set of edges, defined as $\{(v_X, v_{X'}) \mid X' \in \mathsf{occ}(\phi_X)\}$;*

- $r{:}V \to \mathbb{N}$ *is a* ranking*, defined as* $r(v_X) = i + \delta$ *for* i *satisfying* $X \in \mathsf{bnd}(\mathcal{E}\rceil i)$*;*
- $l{:}V \to \mathbb{B}$ *is defined as* $l(v_X) = \top$ *iff* ϕ_X *is conjunctive;*

Quotienting a dependency graph using some suitable equivalence relation, and subsequently converting the resulting dependency graph to a Boolean equation system can result in a significant speed-up for solving an equation system. A recently introduced equivalence relation, tailored to this purpose is *idempotence-identifying bisimulation*, see [10].

Definition 12. *Let* $G = \langle V, \to, r, l\rangle$ *be a dependency graph. A symmetric relation* $R \subseteq V \times V$ *is an* idempotence-identifying bisimulation *if* vRw *implies:*

1. $r(v) = r(w)$*;*
2. $l(v) = l(w)$ *or* $\forall v', w' {\in} V : v \to v' \wedge w \to w' \implies v'Rw'$*;*
3. $\forall v' {\in} V : v \to v' \implies \exists w' {\in} V : w \to w' \wedge v'Rw'$*.*

Two vertices v, w *are said to be* idempotence-identifying bisimilar*, denoted* $v \overset{ii}{\leftrightarrow} w$*, iff there is an idempotence-identifying bisimulation* R*, such that* vRw*.*

The theorem below confirms that a consistent correlation reduces to idempotence-identifying bisimilarity for closed Boolean equation systems in standard recursive form, and *vice versa.* This yields a graph based view on consistent correlations.

Theorem 3. *Let* $\mathcal{E}$ *be a closed Boolean equation system in standard recursive form. Let* $G = \langle V, \to, r, l\rangle$ *be* $\mathcal{E}$*'s dependency graph. For all* $X, X' \in \mathsf{bnd}(\mathcal{E})$*, we have* $X \doteqdot X'$ *iff* $v_X \overset{ii}{\leftrightarrow} v_{X'}$*.*

Proof. Follows from a straightforward application of the definitions. □

As a consequence of the above theorem, we find the below decidability result for consistent correlations. This follows from the decidability of $\overset{ii}{\leftrightarrow}$, see [10].

Proposition 1. *Deciding* $X \doteqdot X'$ *requires* $\mathcal{O}(|\mathcal{E}| \log |\mathsf{bnd}(\mathcal{E})|)$ *for Boolean equation systems in standard form.*

Observe that the complexity of the extension to *open* Boolean equation systems, not in recursive form, does not come with a penalty. The complexity of deciding $\doteqdot$ on Boolean equation systems that are not in standard form is still open, but the problem is likely to be NP-complete.

5 Model Checking

We next show that bisimilarity on processes induces consistent correlations on the equation systems that encode model checking problems. This implies that the concept of a consistent correlation does not discriminate beyond bisimilarity on processes, which would otherwise impair its applicability.

We use *linear process equations* (LPEs), see e.g. [6], to *symbolically* represent (possibly infinite) labelled transition systems. An LPE typically models the state of a system by means of a finite vector of sorted formal parameters; the behaviour is described by a finite set of condition-action-effect rules. Interleaving parallelism, (infinite) non-determinism and other means of composing systems can be mapped losslessly onto LPEs, often fully automatically.

Definition 13. *A* linear process equation *is an equation of the form*

$$L(d_L{:}D_L) = \sum\{\sum_{e_{\mathtt{a}}:E_{\mathtt{a}}} c_{\mathtt{a}}(d_L, e_{\mathtt{a}}) \to \mathtt{a}(f_{\mathtt{a}}(d_L, e_{\mathtt{a}})) \cdot L(g_{\mathtt{a}}(d_L, e_{\mathtt{a}})) \mid \mathtt{a} \in \mathcal{A}ct\}$$

here, we have $f_{\mathtt{a}}{:}D_L \times E_{\mathtt{a}} \to D_{\mathtt{a}}$, $g_{\mathtt{a}}{:}D_L \times E_{\mathtt{a}} \to D_L$ and $c_{\mathtt{a}}{:}D_L \times E_{\mathtt{a}} \to B$ for each action label $\mathtt{a}{:}D_{\mathtt{a}}$ taken from a finite set $\mathcal{A}ct$ of action labels. The interpretation *of L, given a data environment ρ, is a* labelled transition system $\langle \mathbb{D}_L, A, \to\rangle$*:*

- $A = \{\mathtt{a}(v_{\mathtt{a}}) \mid \mathtt{a} \in \mathcal{A}ct, v_{\mathtt{a}} \in \mathbb{D}_{\mathtt{a}}\}$ *is the set of* actions*;*
- $\to \subseteq \mathbb{D}_L \times A \times \mathbb{D}_L$ *is the set of transitions, where $v \xrightarrow{\mathtt{a}(v_{\mathtt{a}})} w$ iff for some $u \in \mathbb{E}_{\mathtt{a}}$:*
  - $[\![f_{\mathtt{a}}(d_L, e_{\mathtt{a}})]\!]\rho[v/d_L, u/e_{\mathtt{a}}] = v_{\mathtt{a}}$ *and* $[\![g_{\mathtt{a}}(d_L, e_{\mathtt{a}})]\!]\rho[v/d_L, u/e_{\mathtt{a}}] = w$*,*
  - $[\![c_{\mathtt{a}}(d_L, e_{\mathtt{a}})]\!]\rho[v/d_L, u/e_{\mathtt{a}}]$ *evaluates to true.*

Throughout this section, whenever we refer to an LPE L we implicitly mean the LPE as given by Def. 13, with d_L of being the state parameter of sort D_L of process L. An LPE L compactly expresses that in a state, represented by parameter d_L, whenever condition $c_{\mathtt{a}}(d_L, e_{\mathtt{a}})$ holds (for some non-deterministically chosen value for variable $e_{\mathtt{a}}$), then action $\mathtt{a}$ carrying data parameter $f_{\mathtt{a}}(d_L, e_{\mathtt{a}})$ can be executed, effectively changing the global state to $g_{\mathtt{a}}(d_L, e_{\mathtt{a}})$.

Definition 14. *Let $\langle \mathbb{D}_L, A, \to\rangle$ be the transition system underlying LPE L. A symmetric relation $R \subseteq \mathbb{D}_L \times \mathbb{D}_L$ is a* bisimulation relation*, if vRw implies:*

$$\forall v' \in \mathbb{D}_L, \mathtt{a}(v_{\mathtt{a}}) \in A : v \xrightarrow{\mathtt{a}(v_{\mathtt{a}})} v' \implies \exists w' \in \mathbb{D}_L : w \xrightarrow{\mathtt{a}(v_{\mathtt{a}})} w' \wedge v'Rw'$$

We say states v, w are bisimilar, denoted $v \,\underline{\leftrightarrow}\, w$ iff there is some bisimulation relation R such that vRw.

Model checking is concerned with checking whether a modal property holds for a given system or not. The *first-order modal μ-calculus* (μ-calculus for short) of [6] is a language for expressing such properties. It permits the use of data to capture the essential data-influences in a system's behaviour.

Definition 15. *The grammar of the μ-calculus is given by the following rules:*

$$\begin{aligned} \phi, \psi &::= b \mid \bar{X}(e) \mid \phi \wedge \psi \mid \phi \vee \psi \mid \forall d{:}D.\ \phi \mid \exists d{:}D.\ \phi \mid \langle\alpha\rangle\phi \mid [\alpha]\phi \mid \\ &\qquad (\nu \bar{X}(d_{\bar{X}}{:}D_{\bar{X}} := e).\ \phi) \mid (\mu \bar{X}(d_{\bar{X}}{:}D_{\bar{X}} := e).\ \phi) \\ \alpha, \beta &::= b \mid \mathtt{a}(e') \mid \neg\alpha \mid \alpha \wedge \beta \mid \forall d{:}D.\ \alpha \end{aligned}$$

here, b is a Boolean expression, $\bar{X}$ is a recursion variable of sort $D_{\bar{X}} \to 2^{D_L}$ taken from some sufficiently large set $\bar{\mathcal{P}}$ of recursion variables, $d_{\bar{X}}$ and e are a data variable and data expression of sort $D_{\bar{X}}$, and e' is of sort $D_{\mathtt{a}}$.

μ-Calculus formulae are interpreted in the context of a labelled transition system induced by LPE L. In particular, the action formulae α describe a (possibly infinite) set of actions. For details, we refer to [6]. The global model checking problem $L \models \varPhi$ and the local model checking problem $L(e) \models \varPhi$, where e is an initial value for L and $\varPhi$ is a μ-calculus formula, can be translated to the

Table 1. Translation scheme for encoding the problem $L \models \sigma\bar{X}(d_{\bar{X}}{:}D_{\bar{X}} := e).\ \psi$, for LPE L and normalised formula $\sigma\bar{X}(d_{\bar{X}}{:}D_{\bar{X}} := e).\ \psi$, into an equation system. Recall that parameter d_L of sort D_L originates from L.

$\mathbf{E}_L(b)$	$\hat{=}$	ϵ	
$\mathbf{E}_L(\bar{X}(e))$	$\hat{=}$	ϵ	
$\mathbf{E}_L(\phi \oplus \psi)$	$\hat{=}$	$\mathbf{E}_L(\phi)\ \mathbf{E}_L(\psi)$	for $\oplus \in \{\land, \lor\}$
$\mathbf{E}_L(\mathsf{Q}\, d{:}D.\phi)$	$\hat{=}$	$\mathbf{E}_L(\phi)$	for $\mathsf{Q} \in \{\forall, \exists\}$
$\mathbf{E}_L([\alpha]\phi)$	$\hat{=}$	$\mathbf{E}_L(\phi)$	
$\mathbf{E}_L(\langle\alpha\rangle\phi)$	$\hat{=}$	$\mathbf{E}_L(\phi)$	
$\mathbf{E}_L(\sigma\bar{X}(d_{\bar{X}}{:}D_{\bar{X}} := e).\ \psi)$	$\hat{=}$	$(\sigma X(d_L{:}D_L, d_{\bar{X}}{:}D_{\bar{X}}) = \mathbf{RHS}_L(\psi))\ \mathbf{E}_L(\psi)$	
$\mathbf{RHS}_L(b)$	$\hat{=}$	b	
$\mathbf{RHS}_L(\bar{X}(e))$	$\hat{=}$	$X(d_L, e)$	
$\mathbf{RHS}_L(\phi \oplus \psi)$	$\hat{=}$	$\mathbf{RHS}_L(\phi) \oplus \mathbf{RHS}_L(\psi)$	for $\oplus \in \{\land, \lor\}$
$\mathbf{RHS}_L(\mathsf{Q}\, d{:}D.\phi)$	$\hat{=}$	$\mathsf{Q}\, d{:}D.\ \mathbf{RHS}_L(\phi)$	for $\mathsf{Q} \in \{\forall, \exists\}$
$\mathbf{RHS}_L([\alpha]\phi)$	$\hat{=}$	$\bigwedge_{\mathsf{a}\in\mathcal{A}ct}\ \forall e_{\mathsf{a}}{:}D_{\mathsf{a}}.\ (c_{\mathsf{a}}(d_L, e_{\mathsf{a}}) \land \mathsf{match}(\mathsf{a}(f_{\mathsf{a}}(d_L, e_{\mathsf{a}})), \alpha))$ $\implies (\mathbf{RHS}_L(\phi)[g_{\mathsf{a}}(d_L, e_{\mathsf{a}})/d_L])$	
$\mathbf{RHS}_L(\langle\alpha\rangle\phi)$	$\hat{=}$	$\bigvee_{\mathsf{a}\in\mathcal{A}ct}\ \exists e_{\mathsf{a}}{:}D_{\mathsf{a}}.\ (c_{\mathsf{a}}(d_L, e_{\mathsf{a}}) \land \mathsf{match}(\mathsf{a}(f_{\mathsf{a}}(d_L, e_{\mathsf{a}})), \alpha)$ $\land\ (\mathbf{RHS}_L(\phi)[g_{\mathsf{a}}(d_L, e_{\mathsf{a}})/d_L]))$	
$\mathbf{RHS}_L(\sigma\bar{X}(d_{\bar{X}}{:}D_{\bar{X}} := e).\ \phi)$	$\hat{=}$	$X(d_L, e)$	
$\mathsf{match}(\mathsf{a}(v), b)$	$\hat{=}$	b	
$\mathsf{match}(\mathsf{a}(v), \mathsf{a}'(e'))$	$\hat{=}$	$(v = e') \land (\mathsf{a} = \mathsf{a}')$	
$\mathsf{match}(\mathsf{a}(v), \neg\alpha)$	$\hat{=}$	$\neg\mathsf{match}(\mathsf{a}(v), \alpha)$	
$\mathsf{match}(\mathsf{a}(v), \alpha \land \beta)$	$\hat{=}$	$\mathsf{match}(\mathsf{a}(v), \alpha) \land \mathsf{match}(\mathsf{a}(v), \beta)$	
$\mathsf{match}(\mathsf{a}(v), \forall d{:}D.\ \alpha)$	$\hat{=}$	$\forall d{:}D.\ \mathsf{match}(\mathsf{a}(v), \alpha)$	

problem of solving the equation system $\mathbf{E}_L(\Phi)$, see e.g. [8]. Table 1 contains the translation; it assumes that the input formula $\sigma\bar{X}(d_{\bar{X}}{:}D_{\bar{X}} := e).\ \psi$ is *normalised*, i.e., none of its subformulae of the form $\sigma'\bar{Y}(d_{\bar{Y}}{:}D_{\bar{Y}} := e').\ \psi'$ contain unbound data variables. Rules for normalising can be found in [12].

Lemma 1. *Let ϕ be a normalised μ-calculus formula, ranging over a set of recursion variables $\bar{P} \subseteq \bar{\mathcal{P}}$. Assume that $R \subseteq \mathbb{D}_L \times \mathbb{D}_L$ is a bisimulation relation on the transition system induced by L. Define $S \subseteq \mathsf{sig}(P) \times \mathsf{sig}(P)$ as $(v, w)_X S (v', w)_X$ for all $(v, w), (v', w) \in \mathbb{D}_L \times \mathbb{D}_{\bar{X}}$, for $\bar{X} \in \bar{P}$ and vRv'.*

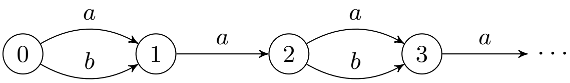
$$\forall v, v' \in \mathbb{D}_L, \theta \in \Theta_S, \varepsilon :\ vRv' \implies [\![\mathbf{RHS}_L(\phi)]\!]\theta\varepsilon[v/d_L] = [\![\mathbf{RHS}_L(\phi)]\!]\theta\varepsilon[v'/d_L]$$

Proof. The proof proceeds by induction on the structure of ϕ. □

Theorem 4. *Let ϕ be a closed normalised μ-calculus formula ranging over a set of recursion variables $\bar{P} \subseteq \bar{\mathcal{P}}$. Let L be an LPE. We have $(v, w)_X \div (v', w)_X$ for all $(v, w), (v', w) \in \mathbb{D}_L \times \mathbb{D}_{\bar{X}}$, $\bar{X} \in \bar{P}$, satisfying $v \underline{\leftrightarrow} v'$.*

Proof. The proof follows instantly from Lemma 1 and the definition of $\mathbf{E}$. □

Example 3. Let $L(n{:}N) = \mathsf{a} \cdot L(n+1) + \mathsf{even}(n) \rightarrow \mathsf{b} \cdot L(n+1)$ be an LPE, encoding the infinite-state process that can always perform an a-action, and at each even moment, it can perform a b-action, see the figure below.



Clearly, for $v, w \in \mathbb{N}$, we have $v \underline{\leftrightarrow} w$ iff $v \equiv_2 w$. Verifying that action b can be executed infinitely often, i.e., encoding the formula $\phi \equiv \nu\bar{X}.\mu\bar{Y}.(\langle \mathsf{b} \rangle \bar{X} \vee \langle \mathsf{a} \rangle \bar{Y})$ results in the following equation system $\mathbf{E}_L(\phi)$:

$$\big(\nu X(n{:}N) = Y(n)\big)\ \big(\mu Y(n{:}N) = (\mathsf{even}(n) \wedge X(n+1)) \vee Y(n+1)\big)$$

It follows immediately that bisimilarity induces a consistent correlation on $\mathbf{E}_L(\phi)$, i.e., we have $v_X \doteqdot w_X$ and $v_Y \doteqdot w_Y$ for $v \equiv_2 w$, again for $v, w \in \mathbb{N}$. Using Theorem 2, we have $v_X \doteqdot \mathsf{even}(v)_{X'}$ and $v_Y \doteqdot \mathsf{even}(v)_{Y'}$, for all $v \in \mathbb{N}$, where X', Y' are given through the following equation system:

$$\big(\nu X'(b{:}B) = Y'(b)\big)\ \big(\mu Y'(b{:}B) = (b \wedge X'(\neg b)) \vee Y'(\neg b)\big)$$

Solving the latter via a transformation to a Boolean equation system results in $X'(\mathsf{even}(v)) = Y'(\mathsf{even}(v)) = X(v) = Y(v) = \top$ for all $v \in \mathbb{N}$. □

Observe that the reverse of Theorem 4 does not hold: for instance, $\mathbf{E}_L(\nu\bar{X}.\langle a \rangle \bar{X})$ would have $v_X \doteqdot v'_X$ for all $v, v' \in \mathbb{N}$, but, clearly, not $v \underline{\leftrightarrow} v'$ in L.

6 Applications

We finish with two manipulations from the literature whose correctness follows immediately from the existence of consistent correlations. In a similar vein, the correctness of the static analysis techniques of [14] follow.

6.1 Enumeration

Enumeration of finite data sorts is used to reduce the complexity of the right-hand sides of an equation system. It achieves so by instantiating a data parameter of a finite sort with the basic elements of that sort, introducing one new equation per basic element. For the purpose of readability, we assume that each equation in an equation system consists of two (complex) data sorts: sorts D representing finite data sorts we wish to enumerate, and sorts E representing the possibly infinite data sorts. The scheme is rather straightforward, see Table 2; its correctness proof on the other hand, is rather arduous, spanning several pages, see [2]. The below proposition paraphrases Theorem 1 of [2].

Proposition 2. *For all closed equation systems $\mathcal{E}$ and all $P \subseteq \mathsf{bnd}(\mathcal{E})$, we have for all $X \in P$, $(\llbracket v \rrbracket, w)_X \doteqdot w_{\tilde{X}_v}$, and for all $X \notin P$, $(\llbracket v \rrbracket, w)_X \doteqdot (\llbracket v \rrbracket, w)_{\tilde{X}}$.*

Table 2. The enumeration scheme for finite data sorts D, and $P \subseteq \mathcal{P}$

$$
\begin{array}{l}
\mathsf{Inst}_P(\epsilon) \mathrel{\hat{=}} \epsilon \\
\mathsf{Inst}_P((\sigma X(d_X{:}D_X, e_X{:}E_X) = \phi_X)\ \mathcal{E}) \mathrel{\hat{=}} \\
\qquad \begin{cases} \{(\sigma \tilde{X}_v(e_X{:}E_X) = \mathsf{Sub}_P(\phi_X[v/d_X])) \mid v{\in}D_X\}\ \mathsf{Inst}_P(\mathcal{E}) & \text{if } X{\in}P \\ (\sigma \tilde{X}(d_X{:}D_X, e_X{:}E_X) = \mathsf{Sub}_P(\phi))\ \mathsf{Inst}_P(\mathcal{E}) & \text{otherwise} \end{cases}
\end{array}
$$

where

$$
\begin{array}{lll}
\mathsf{Sub}_P(b) & \mathrel{\hat{=}} b & \\
\mathsf{Sub}_P(X(d,e)) & \mathrel{\hat{=}} \begin{cases} \bigvee_{u \in D_X}(u = d \wedge \tilde{X}_u(e)) \\ \tilde{X}(d,e) \end{cases} & \begin{array}{l} \text{if } X{\in}P \\ \text{otherwise} \end{array} \\
\mathsf{Sub}_P(\phi \oplus \psi) & \mathrel{\hat{=}} \mathsf{Sub}_P(\phi) \oplus \mathsf{Sub}_P(\psi) & \text{for } \oplus \in \{\wedge, \vee\} \\
\mathsf{Sub}_P(\mathsf{Q}d{:}D.\ \phi) & \mathrel{\hat{=}} \mathsf{Q}d{:}D.\ \mathsf{Sub}_P(\phi) & \text{for } \mathsf{Q} \in \{\forall, \exists\}
\end{array}
$$

Proof. The correctness follows from the observation that the relation R, defined as $(\llbracket v \rrbracket, w)_X R w_{\tilde{X}_v}$, for all $X \in P$, and $(\llbracket v \rrbracket, w)_X R(\llbracket v \rrbracket, w)_{\tilde{X}}$ for all $X \notin P$, is a consistent correlation. The fact that R is a consistent correlation follows from an induction on the structure of the predicate formulae, showing that for all ϕ and all $\theta \in \Theta_R$, ε, $\llbracket \phi \rrbracket \theta\varepsilon[\llbracket v \rrbracket/d, w/e] = \llbracket \mathsf{Sub}_P(\phi[v/d_X]) \rrbracket \theta\varepsilon[w/e]$. □

6.2 Global Invariants

The second manipulation that we consider is the use of *global invariants*, see [15], for strengthening equation systems. Let Pred denote the predicate variable-free fragment of the predicate formulae, called *simple formulae.*

Definition 16. *The function $f{:}V \to \mathsf{Pred}$ is said to be a* global invariant *for an equation system $\mathcal{E}$ iff $\mathsf{bnd}(\mathcal{E}) \subseteq V$ and for each $X \in \mathsf{bnd}(\mathcal{E})$, we have for all θ, ε:*

$$\llbracket f(X) \wedge \phi_X \rrbracket \theta\varepsilon = \llbracket (f(X) \wedge \phi_X) \left[_{Y \in V} \lambda d_Y.(f(Y) \wedge Y(d_Y))/Y \right] \rrbracket \theta\varepsilon$$

Note that $\psi\left[_{Y \in V}\ \lambda d_Y.\zeta/Y\right]$ stands for a simultaneous syntactic substitution of the functional $\lambda d_Y.\zeta$ for every $Y \in V$ in ψ. For details, we refer to [15]. The invariance condition basically states that the right-hand sides of all equations should be insensitive to strengthening all predicate variable occurrences with their corresponding simple formula.

Definition 17. *Let $\mathcal{E}$ be an equation system and let $f{:}V \to \mathsf{Pred}$ be a global invariant for $\mathcal{E}$. The equation system $\mathit{Apply}\ (f, \mathcal{E})$ is defined as follows:*

$$
\begin{array}{ll}
\mathit{Apply}\ (f, \epsilon) & \mathrel{\hat{=}} \epsilon \\
\mathit{Apply}\ (f, (\sigma X(d_X{:}D_X) = \phi_X)\ \mathcal{E}_0) & \mathrel{\hat{=}} (\sigma \tilde{X}(d_X{:}D_X) = f(X) \wedge \phi_{\tilde{X}})\ \mathit{Apply}\ (f, \mathcal{E}_0)
\end{array}
$$

where $\phi_{\tilde{X}}$ is the formula ϕ_X in which all $Y \in \mathsf{occ}(\phi_X)$ are replaced by $\tilde{Y}$.

The following proposition paraphrases Theorem 35 of [15].

Proposition 3. *Let $\mathcal{E}$ be a closed equation system and let f be a global invariant for $\mathcal{E}$. Then $v_X \doteqdot v_{\tilde{X}}$ for all $X \in \mathsf{bnd}(\mathcal{E})$ and $\tilde{X} \in \mathsf{bnd}(\mathit{Apply}\ (f, \mathcal{E}))$ and all v satisfying $[\![\lambda d_X . f(X)]\!](v) = \top$.*

Proof. The relation R, defined as $v_X R v_{\tilde{X}}$ for all v such that $[\![f(X)]\!]\epsilon[v/d_X]$ holds, is a consistent correlation. This follows from the observation that, by definition of a global invariant, each pair of formulae $\phi_X, \phi_{\tilde{X}}$ depends, semantically, only on instances that satisfy the global invariant, and, thereby are related via R. □

7 Concluding Remarks

We introduced the concept of a *consistent correlation* in the setting of equation systems, and showed that it provides a natural and direct method for proving the correctness of manipulations in this setting; in addition, we studied its relation to bisimulations on processes and dependency graphs for equation systems.

Several issues are still open for investigation. First, the exact complexity for computing a consistent correlation for arbitrary (Boolean) equation systems is still unknown. Second, our theory of consistent correlations is phrased entirely in terms of semantics. It is certainly interesting to devise a syntax based theory that is rooted in the same concepts. We conjecture that it suffices to add the following deduction rule to the standard axioms for negation-free propositional logic, to obtain a sound and complete set of rules for Boolean equation systems:

$$\frac{\Gamma, X \doteqdot Y \vdash \phi_X \doteqdot \phi_Y \qquad X, Y \in \mathsf{bnd}(\mathcal{E}\rceil i)}{\Gamma \vdash X \doteqdot Y}$$

Here, $\doteqdot$ is lifted to propositional formulae and Γ serves as a context (i.e., assumptions). Third, it is likely that the problem of computing a consistent correlation of an equation system can itself be encoded in terms of an equation system, in the spirit of [1], although it is unclear whether such an encoding will be worthwhile from a practical viewpoint. Finally, it is interesting to see whether consistent correlations can play a role in proving that the static analysis techniques outlined in [16] carry over to the setting of equation systems.

Acknowledgements. The author would like to thank Jeroen Keiren and Michel Reniers for discussions and comments on earlier drafts of this paper.

References

1. Chen, T., Ploeger, B., van de Pol, J., Willemse, T.A.C.: Equivalence checking for infinite systems using parameterized Boolean equation systems. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 120–135. Springer, Heidelberg (2007)
2. van Dam, A., Ploeger, B., Willemse, T.A.C.: Instantiation for parameterised Boolean equation systems. In: Fitzgerald, J.S., Haxthausen, A.E., Yenigün, H. (eds.) ICTAC 2008. LNCS, vol. 5160, pp. 440–454. Springer, Heidelberg (2008)

3. Gallardo, M., Joubert, C., Merino, P.: Implementing influence analysis using parameterised Boolean equation systems. In: ISOLA. IEEE Computer Society Press, Los Alamitos (November 2006)
4. Garavel, H., Mateescu, R., Lang, F., Serwe, W.: CADP 2006: A toolbox for the construction and analysis of distributed processes. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 158–163. Springer, Heidelberg (2007)
5. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
6. 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)
7. Groote, J.F., Mathijssen, A.H.J., Reniers, M.A., Usenko, Y.S., Weerdenburg, M.J.v.: Analysis of distributed systems with mcrl2. In: Process Algebra for Parallel and Distributed Processing, pp. 99–128. Chapman Hall, Boca Raton (2009)
8. Groote, J.F., Willemse, T.A.C.: Model-checking processes with data. Sci. Comput. Program 56(3), 251–273 (2005)
9. Groote, J.F., Willemse, T.A.C.: Parameterised Boolean equation systems. Theor. Comput. Sci. 343(3), 332–369 (2005)
10. Keiren, J., Willemse, T.A.C.: Bisimulation minimisation of Boolean equation systems. In: HVC 2009. LNCS, Springer, Heidelberg (2009)
11. Mader, A.: Verification of Modal Properties Using Boolean Equation Systems. PhD thesis, Technische Universität München (1997)
12. Mateescu, R.: Vérification des propriétés temporelles des programmes parallèles. PhD thesis, Institut National Polytechnique de Grenoble (1998)
13. Mateescu, R., Thivolle, D.: A model checking language for concurrent value-passing systems. In: Cuéllar, J., Maibaum, T.S.E., Sere, K. (eds.) FM 2008. LNCS, vol. 5014, pp. 148–164. Springer, Heidelberg (2008)
14. Orzan, S., Wesselink, J.W., Willemse, T.A.C.: Static analysis techniques for parameterised Boolean equation systems. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 230–245. Springer, Heidelberg (2009)
15. Orzan, S.M., Willemse, T.A.C.: Invariants for parameterised Boolean equation systems. Theor. Comp. Sci. 411(11-13), 1338–1371 (2010)
16. van de Pol, J., Timmer, M.: State space reduction of linear processes using control flow reconstruction. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 54–68. Springer, Heidelberg (2009)
17. 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)

Author Index