Department of Computing, Imperial College London

Technical Report 2006/10

Preliminary Proceedings

13th International Workshop on
Expressiveness in Concurrency

EXPRESS'06

Bonn, Germany

26 August 2006

Editors:

Roberto Amadio
Iain Phillips

Contents

Preface

The EXPRESS workshops aim at bringing together researchers interested in the relations between various formal systems in computer science, in particular in the field of concurrency. More specifically, they focus on the comparison between programming concepts (such as concurrent, functional, imperative, logic and object-oriented programming) and between mathematical models of computation (such as process algebras, Petri nets, event structures, modal logics, rewrite systems etc.) on the basis of their relative expressive power.

The EXPRESS workshops were originally held as meetings of the HCM project EXPRESS, which was active with the same focus from January 1994 until December 1997. The first three workshops were held in Amsterdam (1994, chaired by Frits Vaandrager), Tarquinia (1995, chaired by Rocco De Nicola) and Dagstuhl (1996, co-chaired by Ursula Goltz and Rocco De Nicola). The workshop in 1997, which took place in Santa Margherita Ligure and was co-chaired by Catuscia Palamidessi and Joachim Parrow, was organized as a conference with a call for papers and a significant attendance from outside the project. As of 1998 (so, also this year), the workshops are held as satellite workshops of the CONCUR conferences. In 1998, this was in Nice, co-chaired by Ilaria Castellani and Catuscia Palamidessi, in 1999 in Eindhoven, co-chaired by Ilaria Castellani and Björn Victor, in 2000 at Pennsylvania State University, co-chaired by Luca Aceto and Björn Victor, in 2001 at BRICS, Aalborg University, co-chaired by Luca Aceto and Prakash Panangaden, in 2002 in Brno, co-chaired by Uwe Nestmann and Prakash Panangaden, in 2003 in Marseille, co-chaired by Flavio Corradini and Uwe Nestmann, in 2004 in London, co-chaired by Jos Baeten and Flavio Corradini, and finally in 2005 in San Francisco, co-chaired by Jos Baeten and Iain Phillips.

This year, in response to the call for papers, we received 24 submissions. The programme committee selected nine of these for presentation at the workshop. Two of these were short papers that do not appear in the proceedings. In addition, the workshop had two invited presentations, by Robin Milner (invited jointly with the Infinity and SOS workshops), and by Hagen Völzer. Abstracts for these talks appear in these preliminary proceedings. We would like to thank the authors of the submitted short and full papers, the invited speakers, the members of the programme committee and their subreferees for their contribution to both the meeting and this volume. We thank the CONCUR organising committee for hosting EXPRESS'06, in particular the workshop organiser Marcus Größer and CONCUR programme committee co-chairs Christel Baier and Holger Hermanns. We are grateful to Marcus Größer for arranging the printing of these preliminary proceedings. Michael Mislove helped with style files, Erik Luit provided invaluable advice on installing Cyberchair, and the Imperial College Department of Computing Computing Support Group (and especially Andy Davies and Duncan White) helped with setting up the webpage `http://www.doc.ic.ac.uk/express06`. The final proceedings will appear in the Electronic Notes in Theoretical Computer Science (ENTCS) series, and will become available electronically at Elsevier Science Publisher's website `http://www.elsevier.nl/locate/entcs`. We are grateful to

ENTCS for their continuing support, in particular to Michael Mislove, Managing Editor of the ENTCS series.

The editors

Roberto Amadio (Université Paris 7)
Iain Phillips (Imperial College London)

EXPRESS 2006 Programme Committee

Robert Amadio
Michele Bugliese
Nadia Busi
Sibylle Fröschle
Antonin Kucera
Bas Luttik
Michael Mislove
Uwe Nestmann
Joel Ouaknine
Catuscia Palamidessi
Iain Phillips
Philippe Schnoebelen
Pawel Sobocinski
Mariëlle Stoelinga

EXPRESS 2006 Subreferees

Martin Berger
Stefan Blom
Johannes Borgström
Roberto Bruni
Troels C. Damgaard
Søren Debois
Yuxin Deng
Susan Eisenbach
Maurizio Gabbrielli
Bartek Klin
Sławomir Lasota
Axel Legay
Damiano Macedonio
Mikkel Nygaard
Luca Padovani
Prakash Panangaden
G. Michele Pinna
Davide Sangiorgi
Ana Sokolova
Martin Steffen
Nikola Trcka
Daniele Varacca
Cristian Versari
James Worrell
Peng Wu
Gianluigi Zavattaro

When a system is fairly correct

Hagen Völzer[1,2]

Institute for Theoretical Computer Science
Lübeck University
Germany

Abstract

We give an overview over recent work on fairness in reactive and concurrent systems, including an abstract characterisation of fairness. We also derive a notion of a *fairly correct* system and sketch its application.

Keywords: Fairness, liveness, temporal properties, verification, model checking

Extended Abstract

Fairness is a convenient and popular tool when modelling and specifying concurrent systems. A large variety of fairness notions exists in the literature. Among them, we find well-known notions such as *weak fairness* (*justice*) [12], *strong fairness* (*compassion*) [12], and *extreme fairness* [15] and less-known notions such as ∞-*fairness* [4], α-*fairness* [13], and *hyperfairness* [3,11,17]. A fairness notion is often meant to represent a particular phenomenon. Phenomena expressed by fairness assumptions include progress of individual processes, general environment behaviour, behaviour of probabilistic choice, impartiality of arbiters and schedulers, and partial synchrony. Many fairness notions are geared to a particular application or specification language. Overviews over fairness can be found in [9,6,12]. More recent studies on fairness include [7,8,11,18].

In contrast to *safety* and *liveness*, which were characterised by Lamport [10] and Alpern and Schneider [1], there was no fully satisfactory abstract characterisation of fairness. However, Apt, Francez, and Katz [2] gave some criteria that must be met by any fairness assumption. Following Lamport [11], we think that their most important criterion is that a fairness assumption must be *machine closed* with respect to the safety property defined by the transition system under consideration.

[1] This extended abstract is based on joint work with Daniele Varacca, Imperial College London, UK and Ekkart Kindler, Paderborn University, Germany

[2] Email: voelzer@tcs.uni-luebeck.de

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

This, basically, means that fairness is imposed in such a way to the transition system that the system 'cannot paint itself into a corner' [2]; i. e. , whatever the system does, it is possible to continue in such a way that the fairness assumption is met. While machine closedness is necessary for a property to be a fairness property, it does not exclude some properties that are intuitively not fairness properties.

We propose (together with Varacca and Kindler [18]) a definition of fairness that refines machine-closure and excludes properties that are intuitively not fairness properties. One characterisation says: A fairness property with respect to a system is a property that contains a property of the form 'If each prefix of a run can be extended to satisfy Q, then that run has always eventually a prefix satisfying Q', where Q is a property of finite runs. We show that fairness is then closed under arbitrary union and countable intersection and that most popular fairness notions satisfy our definition. We give independent characterisations in terms of game theory, language theory, and general topology [18]. It turns out that fairness as we define it coincides with the *co-meager* sets of the natural topology of runs, a subclass of the dense sets. This shows that our characterisation of fairness is in line with the definitions of safety and liveness given by Lamport [10] and Alpern and Schneider [1] since safety properties are the closed sets and liveness properties are the dense sets of that topology.

A co-meager set is a 'large' set in a topological sense. This gives rise to a notion of a 'fairly correct' system [16]: a system is *fairly correct* if its specification is a large set relative to the set of all runs of the system, i.e., most runs of the system satisfy the specification. Equivalently, a system is fairly correct if there exists a fairness assumption under which it is correct. Many distributed, especially fault-tolerant, systems are only fairly correct with respect to their actual specification since often, fully correct solutions are too expensive or do not exist [5].

Another natural way to formalise 'large set' is to mean *probabilistically large*, i.e., a set of measure 1 for a given probability measure. Note that this notion needs a concrete probability measure, which may be hard to justify for a given system. The notions of probabilistic and topological largeness share many properties. A classic mathematical text book [14] is devoted to study their similarities and differences. Although similar, these notions do not coincide in general—in fact, even for the most straightforward probability measure on the set of runs, there are topologically large sets that have probability 0. However, it turns out [16] that the two notions coincide for bounded Borel measures on finite state systems.

It follows that fair correctness of a finite system is decidable and can be checked with the same complexity as usual correctness for LTL and Büchi automata specifications. However, in contrast to usual correctness, for fair correctness, it is not necessary to specify any fairness assumption explicitly.

References

[1] Alpern, B. and F. B. Schneider, *Defining liveness*, Information Processing Letters **21** (1985), pp. 181–185.

[2] Apt, K. R., N. Francez and S. Katz, *Appraising fairness in languages for distributed programming*, Distributed Computing **2** (1988), pp. 226–241.

[3] Attie, P. C., N. Francez and O. Grumberg, *Fairness and hyperfairness in multi-party interactions*, Distributed Computing **6** (1993), pp. 245–254.

[4] Best, E., *Fairness and conspiracies*, Information Processing Letters **18** (1984), pp. 215–220, erratum ibidem 19:162.

[5] Fich, F. E. and E. Ruppert, *Hundreds of impossibility results for distributed computing.*, Distributed Computing **16** (2003), pp. 121–163.

[6] Francez, N., "Fairness," Springer, 1986.

[7] Joung, Y.-J., *On fairness notions in distributed systems, part I: A characterization of implementability*, Information and Computation **166** (2001), pp. 1–34.

[8] Joung, Y.-J., *On fairness notions in distributed systems, part II: Equivalence-completions and their hierarchies*, Information and Computation **166** (2001), pp. 35–60.

[9] Kwiatkowska, M. Z., *Survey of fairness notions*, Information and Software Technology **31** (1989), pp. 371–386.

[10] Lamport, L., *Formal foundation for specification and verification*, in: M. Alford, J. Ansart, G. Hommel, L. Lamport, B. Liskov, G. Mullery and F. Schneider, editors, *Distributed Systems: Methods and Tools for Specification*, LNCS **190**, Springer-Verlag, 1985 .

[11] Lamport, L., *Fairness and hyperfairness*, Distributed Computing **13** (2000), pp. 239–245.

[12] Lehmann, D. J., A. Pnueli and J. Stavi, *Impartiality, justice and fairness: The ethics of concurrent termination.*, in: S. Even and O. Kariv, editors, *ICALP*, LNCS **115** (1981), pp. 264–277.

[13] Lichtenstein, O., A. Pnueli and L. D. Zuck, *The glory of the past*, in: R. Parikh, editor, *Logic of Programs*, LNCS **193** (1985), pp. 196–218.

[14] Oxtoby, J. C., "Measure and Category. A Survey of the Analogies between Topological and Measure Spaces," Springer-Verlag, 1971.

[15] Pnueli, A., *On the extremely fair treatment of probabilistic algorithms*, in: *Proc. 15th Annual Symposium on Theory of Computing (STOC)* (1983), pp. 278–290.

[16] Varacca, D. and H. Völzer, *Temporal logics and model checking for fairly correct systems*, in: *LICS*, 2006.

[17] Völzer, H., *Refinement-robust fairness*, in: *Proc. CONCUR 2002 – 13th International Conference on Concurrency Theory, Brno, Czech Republic*, LNCS **2421** (2002), pp. 547–561.

[18] Völzer, H., D. Varacca and E. Kindler, *Defining fairness*, in: M. Abadi and L. de Alfaro, editors, *CONCUR*, LNCS **3653** (2005), pp. 458–472.

Fair $\Pi^\star$

Diletta Cacciagrano[1], Flavio Corradini[2]

Dipartimento di Matematica e Informatica
Università degli Studi di Camerino, Italy

Catuscia Palamidessi[3]

INRIA Futurs and LIX École Polytechnique, France

Abstract

In this paper, we define fair computations in the π-calculus [MPW92]. We follow Costa and Stirling's approach for CCS-like languages [CS84,CS87] but exploit a more natural labeling method of process actions to filter out unfair process executions. The new labeling allows us to prove all the significant properties of the original one, such as unicity, persistence and disappearance of labels. It also turns out that the labeled π-calculus is a conservative extension of the standard one. We contrast the existing fair testing [BRV95,NC95] with those that naturally arise by imposing weak and strong fairness as defined by Costa and Stirling. This comparison provides the expressiveness of the various fair testing-based semantics and emphasizes the discriminating power of the one already proposed in the literature.

Key words: Pi-Calculus, Testing Semantics, Strong Fairness, Weak Fairness.

1 Introduction

In the theory and practice of parallel systems, fairness plays an important role when describing the system dynamics. Several notions have been proposed in the literature, as in [CS84,CS87], where Costa and Stirling distinguish between fairness of actions in [CS84] (for a CCS-like language without restriction), and fairness of components in [CS87]. In both cases they distinguish between weak fairness and strong fairness. Weak fairness requires that if an action (a component, resp.) can *almost always* proceed, then it must eventually do so, while strong fairness requires that if an action (a component, resp.) can proceed *infinitely often*, then it must proceed infinitely often. The main ingredients of the theory of fairness in [CS84] and [CS87] are:

* This work was supported by the Investment Funds for Basic Research (MIUR-FIRB) project Laboratory of Interdisciplinary Technologies in Bioinformatics (LITBIO) and by Halley Informatica.

[1] Email: diletta.cacciagrano@unicam.it

[2] Email: flavio.corradini@unicam.it

[3] Email: catuscia@lix.polytechnique.fr

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

- *A labeling method for process terms.* This allows to detect the action performed during a transition and the component responsible for it. Labels are strings in $\{0,1\}^*$, associated systematically with operators and basic actions inside a process. Along a computation, labels are unique and, once a label disappears, it does not reappear in the system anymore (unicity, persistence and disappearance properties).
- *Live actions (components, resp.).* An action (a component, resp.) of a process term is live if it can currently be performed (perform an action, resp.). In a term like $(\nu z)(x(y).\bar{z}w.0 \,|\, z(u).0)$, only action x can be performed while z cannot, momentarily.

In this paper, we adapt to the π-calculus [MPW92] the approach to fairness which has been proposed in [CS84,CS87] for CCS-like languages [Mil89]. A difference with [CS87,CS87] is that our labels are pairs $\langle w, n\rangle \in (\{0,1\}^* \times \mathbb{N})$. The first element, w, represents the position of the component (in the term structure) and depends only on the static operators (parallel and restriction). This element ensures the unicity of a label. The second element, n, provides information about the dynamics of the component, more precisely, it indicates how many actions that component has already executed since the beginning of the computation, and it depends only on the dynamic operator (prefix). This second element serves to ensure the disappearence property of a label. So, we have the unicity and disappearence properties of labels like in [CS84,CS87] but, differently from the latter, we keep separated the information about the static and dynamic operators. We believe that this new labeling method represents more faithfully the structure of a process and makes more intuitive the role of the label in the notion of fairness.

The proposed labeling technique allows to define weak and strong fair computations. On the top of them we introduce must testing semantics [BD95], to obtain the so-called weak fair must semantics and strong fair must semantics. These two fair testing semantics are compared with an existing one in the literature - the fair testing [BRV95,NC95] - that does not need any labeling of actions. We present a comparison between fair testing and weak and strong fair must semantics as well as with standard must testing. This comparison emphasizes the expressiveness of the different fair testing semantics especially for what it concerns fair testing. We show interesting side-effects when the must testing is imposed over weak and strong fair computations. In particular, any strong fair computation is also a weak fair one while it turns out that weak fair must semantics is strictly finer than the strong fair must one.

2 The π-calculus

We now briefly recall the basic notions about the (choiceless) π-calculus. Let $\mathcal{N}$ (ranged over by $x, y, z, \ldots$) be a set of names. The set $\mathcal{P}$ (ranged over by $P, Q, R, \ldots$) of processes is generated by the following grammar:

$$P ::= \; 0 \;\Big|\; x(y).P \;\Big|\; \tau.P \;\Big|\; \bar{x}y.P \;\Big|\; P \,|\, P \;\Big|\; (\nu x)P \;\Big|\; !\, x(y).P$$

The input prefix $y(x).P$, and the restriction $(\nu x)P$, act as name binders for the name x in P. The free names $fn(P)$ and the bound names $bn(P)$ of P are defined

as usual. The set of names of P is defined as $n(P) = fn(P) \cup bn(P)$. Only input guarded terms can be in the scope of the bang operator, but this is not a real shortcoming, since this kind of replicator is as expressive as the full bang operator [HY94].

The operational semantics of processes is given via a labeled transition system, whose states are the process themselves. The labels (ranged over by $\mu, \gamma, \ldots$) "correspond" to prefixes, input xy, output $\bar{x}y$ and tau τ, and to the bound output $\bar{x}(y)$ (which models scope extrusion). If $\mu = xy$ or $\mu = \bar{x}y$ or $\mu = \bar{x}(y)$ we define $sub(\mu) = x$ and $obj(\mu) = y$. The functions fn, bn and n are extended to cope with labels as follows:

$$bn(xy) = \emptyset \qquad bn(\bar{x}(y)) = \{y\} \quad bn(\bar{x}y) = \emptyset \qquad bn(\tau) = \emptyset$$

$$fn(xy) = \{x, y\} \quad fn(\bar{x}(y)) = \{x\} \quad fn(\bar{x}y) = \{x, y\} \quad fn(\tau) = \emptyset$$

The transition relation is given in Table 1. We omit symmetric rules of Par, Com and Close for lake of space. We also assume alpha-conversion to avoid collision of free and bound names.

$$\text{Input } x(y).P \xrightarrow{xz} P\{z/y\}$$

$$\text{Output/Tau } \alpha.P \xrightarrow{\alpha} P \text{ where } \alpha = \bar{x}y \text{ or } \alpha = \tau$$

$$\text{Open } \frac{P \xrightarrow{\bar{x}y} P'}{(\nu y)P \xrightarrow{\bar{x}(y)} P'}\, x \neq y \qquad \text{Res } \frac{P \xrightarrow{\mu} P'}{(\nu y)P \xrightarrow{\mu} (\nu y)P'}\, y \notin n(\mu)$$

$$\text{Par } \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\, bn(\mu) \cap fn(Q) = \emptyset$$

$$\text{Com } \frac{P \xrightarrow{xy} P',\ Q \xrightarrow{\bar{x}y} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \text{Close } \frac{P \xrightarrow{xy} P',\ Q \xrightarrow{\bar{x}(y)} Q'}{P \mid Q \xrightarrow{\tau} (\nu y)(P' \mid Q')}$$

$$\text{Bang } !x(y).P \xrightarrow{xz} P\{z/y\} \mid !x(y).P$$

Table 1
Early operational semantics for $\mathcal{P}$ terms.

Definition 2.1 (*Weak transitions*) Let P and Q be $\mathcal{P}$ processes. Then:

- $P \stackrel{\varepsilon}{\Longrightarrow} Q$ iff $\exists P_0, ..., P_n \in \mathcal{P}$, $n \geq 0$, s.t. $P = P_0 \xrightarrow{\tau} ... \xrightarrow{\tau} P_n = Q$;
- $P \stackrel{\mu}{\Longrightarrow} Q$ iff $\exists P_1, P_2 \in \mathcal{P}$ s.t. $P \stackrel{\varepsilon}{\Longrightarrow} P_1 \xrightarrow{\mu} P_2 \stackrel{\varepsilon}{\Longrightarrow} Q$.

Notation 2.1 For convenience, we write $x(y)$ and $\bar{x}y$ instead of $x(y).0$ and $\bar{x}y.0$, respectively. Furthermore, we write $P \xrightarrow{\mu}$ (respectively $P \stackrel{\mu}{\Longrightarrow}$) to mean that there exists P' such that $P \xrightarrow{\mu} P'$ (respectively $P \stackrel{\mu}{\Longrightarrow} P'$) and we write $P \stackrel{\varepsilon}{\Longrightarrow} \xrightarrow{\mu}$ to mean that there are P' and Q such that $P \stackrel{\varepsilon}{\Longrightarrow} P'$ and $P' \xrightarrow{\mu} Q$.

3 Testing semantics

In this section we briefly summarize the basic definitions behind the testing machinery for the π-calculus.

Definition 3.1 (*Observers*)

- Let $\mathcal{N}' = \mathcal{N} \cup \{\omega\}$ be the set of names, assuming $\omega \notin \mathcal{N}$. By convention $fn(\omega) = \{\omega\}$, $bn(\omega) = \emptyset$ and $sub(\omega) = \omega$. ω is used to report success.
- The set $\mathcal{O}$ (ranged over by $o, o', o'', \ldots$) of observers is defined like $\mathcal{P}$, where the grammar is extended with the production $P ::= \omega.P$.
- The operational semantics of $\mathcal{P}$ is extended to $\mathcal{O}$ by adding $\omega.P \xrightarrow{\omega} P$.

Definition 3.2 (*Experiments*) The set of experiments $\mathcal{E}$ is the set $\{ (P \mid o) \mid P \in \mathcal{P} \text{ and } o \in \mathcal{O}\}$.

Definition 3.3 (*Maximal computations*) Given $P \in \mathcal{P}$ and $o \in \mathcal{O}$, a maximal computation from $P \mid o$ is either an infinite sequence of the form

$$P \mid o = T_0 \xrightarrow{\tau} T_1 \xrightarrow{\tau} T_2 \xrightarrow{\tau} \ldots$$

or a finite sequence of the form

$$P \mid o = T_0 \xrightarrow{\tau} T_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} T_n \not\xrightarrow{\tau} .$$

We are now ready to define must and fair testing semantics.

Definition 3.4 (*Must and Fair Testing Semantics*) Given a process $P \in \mathcal{P}$ and an observer $o \in \mathcal{O}$, define:

- P *must* o if and only if *for every* maximal computation from $P \mid o$
$$P \mid o = T_0 \xrightarrow{\tau} T_1 \xrightarrow{\tau} \ldots T_i \,[\xrightarrow{\tau} \ldots]$$
there exists $i \geq 0$ such that $T_i \xrightarrow{\omega}$;
- P *fair* o if and only if *for every* maximal computation from $P \mid o$
$$P \mid o = T_0 \xrightarrow{\tau} T_1 \xrightarrow{\tau} \ldots T_i \,[\xrightarrow{\tau} \ldots]$$
$T_i \overset{\omega}{\Longrightarrow}$, for every $i \geq 0$.

4 A labeled version of the π-calculus

Fairness imposes that concurrent subprocesses always eventually proceed unless they are deadlocked or have terminated. Such a constraint will affect the behavior of processes. Consider, for instance, the process $P \mid P$, where $P = (\nu a)(!a.\bar{a} \mid \bar{a})$ and the computation $P \mid P \xrightarrow{\tau} P \mid P \xrightarrow{\tau} P \mid P \xrightarrow{\tau} ..$ We can not know whether the computation is fair or not, since we do not know which component (either on the right hand or on the left one of $\mid$), at each reduction step performs a synchronization: we need to distinguish unambiguously actions of a concurrent system and to monitor them along its computations.

For this purpose, we extend to the π-calculus the label-based approach proposed in [CS87]. As explained in the introduction, however, we depart from [CS87] in the way we define the labels. In our case, labels are pairs whose first and second elements

represent, respectively, the position of the component in the term and the number of actions already executed.

Definition 4.1 Let $P \in \mathcal{P}$. Define $L_{\langle s,n\rangle}(P)$, where $s \in \{0,1\}^*$ and $n \in \mathbb{N}$, inductively as follows:

$L_{\langle s,n\rangle}(0) = 0$;
$L_{\langle s,n\rangle}(\mu.P) = \mu_{\langle s,n\rangle}.L_{\langle s,n+1\rangle}(P) \quad (\mu \in \{x(y), \bar{x}y, \tau\})$;
$L_{\langle s,n\rangle}(P \mid Q) = L_{\langle s0,n\rangle}(P) \mid L_{\langle s1,n\rangle}(Q)$;
$L_{\langle s,n\rangle}((\nu x)P) = (\nu x)L_{\langle s,n\rangle}(P)$;
$L_{\langle s,n\rangle}(!x(y).P) = !L_{\langle s,n\rangle}(x(y).P)$.

We proceed by defining $\mathcal{L}(B)$, as the language generated by the grammar

$$B ::= 0 \;\Big|\; L_{\langle s,n\rangle}(\mu.P) \;\Big|\; (\nu x)B \;\Big|\; B \mid B \;\Big|\; !L_{\langle s,n\rangle}(x(y).P)$$

where $s \in \{0,1\}^*$, $n \in \mathbb{N}$, $P \in \mathcal{P}$ and $\mu \in \{x(y), \bar{x}y, \tau\}$.

We now define a binary relation $\Re$ over sets of labels and two functions, *top* and *lab*, allowing to obtain all labels appearing on the top of a labeled process and the whole labels set, respectively.

Definition 4.2 Let $L_1, L_2 \subseteq (\{0,1\}^* \times \mathbb{N})$. We define $L_1 \,\Re\, L_2$ if and only if $\forall \langle s_1, n_1\rangle \in L_1$, $\forall \langle s_2, n_2\rangle \in L_2$, $s_1 \not\leq s_2$ and $s_2 \not\leq s_1$, where $\leq$ is the usual prefix relation between strings.

Definition 4.3 Let $E \in \mathcal{L}(B)$. $top(E)$ and $lab(E)$ are defined by structural induction as follows:

$E = 0 : top(E) = \emptyset$	$lab(E) = \emptyset$
$E = L_{\langle s,n\rangle}(\mu.P) : top(E) = \{\langle s,n\rangle\}$	$lab(E) = \{\langle s,n\rangle\} \cup lab(L_{\langle s,n+1\rangle}(P))$
$E = (\nu x)E' : top(E) = top(E')$	$lab(E) = lab(E')$
$E = E_1 \mid E_2 : top(E) = top(E_1) \cup top(E_2)$	$lab(E) = lab(E_1) \cup lab(E_2)$
$E = !L_{\langle s,n\rangle}(x(y).P) : top(E) = \{\langle s,n\rangle\}$	$lab(E) = lab(L_{\langle s,n\rangle}(x(y).P))$

Now, we are ready to define $\mathcal{P}^e$, the set of labeled π-calculus terms.

Definition 4.4 $\mathcal{P}^e$ is the set $\{E \in \mathcal{L}(B) \mid wf(E)\}$, where $wf(E)$ is defined in Table 2.

It is possible to verify that $\forall E \in \mathcal{P}^e$, $top(E) \subseteq lab(E)$. The intuition behind the definitions of *top* and *lab* in the case of a bang term follows by viewing $!L_{\langle s,n\rangle}(x(y).P)$ as $L_{\langle s,n\rangle}(x(y).(P \mid !x(y).P))$.

The operational semantics of $\mathcal{P}^e$ is similar to the one in Table 1; we simply ignore labels in order to derive a transition. As expected, the only rule that needs attention regards bang processes, because the unfolding generates new components and we must ensure unicity of labels. Since the unfolding puts two components in parallel, we exploit a proper dynamic labeling of the parallel components (Table 3).

To give some more intuition, consider $S = x(y).(z(k) \mid \bar{z}h) \mid f$ and its label version

$$\text{Nil/Prefix/Bang}\quad \frac{E = 0 \vee E = L_{\langle s,n\rangle}(\mu.P) \vee E = !L_{\langle s,n\rangle}(x(y).P))}{wf(E)}$$

$$\text{Res}\quad \frac{wf(E)}{wf((\nu x)E)}$$

$$\text{Par}\quad \frac{wf(E_1),\quad wf(E_2),\quad top(E_1)\ \Re\ top(E_2)}{wf(E_1 \mid E_2)}$$

Table 2
Well Formed Terms.

$$\text{Bang}\quad !x(y).P \xrightarrow{xz} P\{z/y\} \mid !x(y).P$$

$$\text{Bang}(\mathcal{P}^e)\quad !L_{\langle s,n\rangle}(x(y).P) \xrightarrow{xz} L_{\langle s0,n+1\rangle}(P\{z/y\}) \mid !L_{\langle s1,n+1\rangle}(x(y).P)$$

Table 3
Bang Rules.

$S' = x(y)_{\langle 0,0\rangle}.(z(k)_{\langle 00,1\rangle} \mid \bar{z}h_{\langle 01,1\rangle}) \mid f_{\langle 1,0\rangle}$. [4] Prefixes $x(y)$ and f in S are both top level prefixes. For this reason, they get labels of length 1; though the one on the left hand side of the parallel composition has been labeled 0, while the one on the right hand side has been labeled 1, just to distinguish the two prefixes. On the other hand, $z(k)$ and $\bar{z}h$ within the scope of $x(y)$ are both second level prefixes composed in parallel, so that they get 00 and 01 as different parallel subcomponents, respectively. However, as second action of the source component, they have the same index (i.e. 1). The significance of the second element of the labels is, of course, more evident when we consider more sequential processes.

$\mathcal{P}^e$ enjoys closure properties under any renamings σ, since σ does not change labels. Hence, it is closed under the execution of basic actions. Furthermore, no label occurs more than once in a labeled term (*unicity of labels*) and once a label disappears (it happens when the action related to such a label is performed) along a computation, it does not appear in the system anymore (*persistence and disappearance of labels*). As expected, it is also a *conservative extension* of the unlabeled language: denoting by $Unl(E)$ the $\mathcal{P}$ process obtained by removing all labels in $E \in \mathcal{P}^e$, we can prove that $E \xrightarrow{\mu} E'$ implies $Unl(E) \xrightarrow{\mu} Unl(E')$ and $Unl(E) \xrightarrow{\mu} P'$ implies $\exists E' \in \mathcal{P}^e$ such that $E \xrightarrow{\mu} E'$ and $Unl(E') = P'$.

5 Strong and weak fairness

The labeling method proposed in the previous section can be naturally extended over observers and experiments. In the following, labels will be denoted by $v, v_1, v_2, .. \in (\{0,1\}^* \times \mathbb{N})$ for convenience. $\mathcal{O}^e$ (ranged over by $\rho, \rho', ..$) denotes the set of observers, defined like $\mathcal{P}^e$, where the grammar is enriched with the production $B ::= \omega_v.B$ and the operational semantics is obtained extending $\mathcal{P}^e$ semantics with

[4] According to Costa and Stirling, we have: $S' = x(y)_0.(z(k)_{010}.0_{0101} | _{01}\bar{z}h_{011}.0_{0111}) |_\varepsilon f_1.0_{11}$.

the rule $\omega_v.B \xrightarrow{\omega} B$. $\mathcal{E}^e$ denotes the set of labeled experiments in $\mathcal{P}^e$, as expected.

The definition of *live label* is crucial in every fairness notion. Given a labeled experiment $S \in \mathcal{E}^e$, a *live label* is a label associated to a top-level action which can immediately be performed, i.e. either a τ prefix or a input/output prefix able to synchronize. Given an experiment S, its set of live labels associated to initial τ actions is denoted by $Lp(S)$; notice that, by definition of liveness, if S can not perform any τ step then $Lp(S) = \emptyset$. Since $top(S)$ is defined as the set of any labels appearing on the top of S, $Lp(S) \subseteq top(S)$ follows immediately by the definition of live actions. Now, we can formally define two well-known notions of fairness.

Definition 5.1 (*Weak Fair Computations*) Given $S \in \mathcal{E}^e$, a *weak fair computation* from S is a maximal computation,

$$S \equiv S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} S_2 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_i \,[\xrightarrow{\tau} \ldots]$$

where $\forall v \in (\{0,1\}^* \times \mathbb{N})$, $\forall i \geq 0$, $\exists j \geq i$ such that $v \notin Lp(S_j)$.

Definition 5.2 (*Strong Fair Computations*) Given $S \in \mathcal{E}^e$, a *strong fair computation* from S is a maximal computation,

$$S \equiv S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} S_2 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_i \,[\xrightarrow{\tau} \ldots]$$

where $\forall v \in (\{0,1\}^* \times \mathbb{N})$, $\exists i \geq 0$ such that $\forall j \geq i$, $v \notin Lp(S_j)$.

A *weak fair* computation is a maximal computation such that no label becomes live and then keeps on being live forever. A *strong fair* computation is a maximal computation such that no label is live infinitely often, i.e. no label can become live, lose its liveness, become live again, etc. forever. Formally, strong fairness imposes that for every label there is some point beyond which it never becomes live. Any finite computation is strong fair because all the actions, corresponding to live labels, are performed, and the computation stops when there is no reduction at all. Some useful results follow:

i) every strong fair computation is weak fair, but not the vice versa;

ii) for any labeled experiment S there is a strong fair computation out of S.

Consider item (i). To prove the positive result it suffices to notice that a strong fair computation is a special case of weak fair computation. To prove the negative result, consider $S := !a_{v_1} \mid (\nu b)(!b_{v_2}.(\bar{a}_{v_3} \mid \bar{b}_{v_4}) \mid \bar{b}_{v_5}) \mid a_{v_6}.\omega_{v_7}$. It is not difficult to check that there exists a maximal computation from S, along which a_{v_6} is never performed. It is weak fair but not strong fair.

Now consider item (ii). It suffices to prove that $\forall S \in \mathcal{E}^e$ (a) $Lp(S)$ is a finite set and $S \not\xrightarrow{\tau}$ implies $Lp(S) = \emptyset$; (b)$v \in Lp(S)$ implies $\exists S' \in \mathcal{E}^e$ such that $S \xrightarrow{\mu} S'$ and for any S'' such that $S' \stackrel{\varepsilon}{\Longrightarrow} S''$, $v \notin Lp(S'')$; (c) $\exists S' \in \mathcal{E}^e$ such that $S \stackrel{\varepsilon}{\Longrightarrow} S'$, $Lp(S) \cap Lp(S') = \emptyset$ and for any S'' such that $S' \stackrel{\varepsilon}{\Longrightarrow} S''$, $Lp(S) \cap Lp(S'') = \emptyset$.

6 Comparing fair semantics

In this section we provide a comparison among two different notions of fairness and the must semantics. First of all, it is easy to prove that $\forall P \in \mathcal{P}, \forall o \in \mathcal{O}$, P *must* o implies P *fair* o, but not the vice versa: it suffices to consider the process

$P ::= (\nu a)(!a.\bar{a} \mid \bar{a}) \mid \bar{b}$ and the observer $o ::= b.\omega$.

Then, we try to add fairness in the must testing semantics and investigate the resulting semantic relations.

Definition 6.1 (*Strong and Weak Fair Must Semantics*) Let $E \in \mathcal{P}^e$ and $\rho \in \mathcal{O}^e$. Define $E\, sfmust \rho$ ($E\, wfmust \rho$) if and only if *for every* strong (weak) fair computation from $(E \mid \rho)$

$$E \mid \rho = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_i \,[\xrightarrow{\tau} \ldots]$$

there exists $i \geq 0$ such that $S_i \xrightarrow{\omega}$.

6.1 Weak fairness and strong fairness in a must testing scenario

The following proposition states a very interesting result regarding weak and strong fair must semantics. Notice that the positive implication follows by the fact that an unsuccessful strong fair computation from an experiment $E \mid \rho$ is weak fair too. This result seems to go against a well-established notion stating strong fairness a special case of weak fairness. More in details, it is well-known that strong fairness implies weak fairness, in the sense that a strong fair computation is obviously weak fair too. However, this implication is reversed when the must testing semantics is embedded in this fairness scenario: in the case that every weak fair computation from an experiment is successful, then every strong fair computation from the same experiment is successful.

Proposition 6.2 For every $E \in \mathcal{P}^e$ and $\rho \in \mathcal{O}^e$, then $E\, wfmust \rho$ implies $E\, sfmust \rho$, but not the vice versa.

Must semantics imposes the success over any computation from a given experiment; that being so, any action leading to success in a weak fair computation, can be alternatively live and not only a finite number of steps, since its execution is surely forcing to reach the success. It follows that a successful weak fair computation collapses in a successful strong fair computation. To prove the negative result, consider $E := !a_{v_1} \mid (\nu b)(!b_{v_2}.(\bar{a}_{v_3} \mid \bar{b}_{v_4}) \mid \bar{b}_{v_5})$ and $\rho := a_{v_6}.\omega_{v_7}$.

From $E \mid \rho$ there exists a maximal computation along which every live label different from v_6 is performed, while v_6 becomes live, loses its liveness, becomes live again, etc., without being performed: this computation is weak fair by definition and unsuccessful. Notice that v_6 should be always performed in a strong fair computation, determining the success of it.

Theorem 6.3 shows some interesting results by comparing weak/strong fair must and must semantics.

Theorem 6.3 For every $E \in \mathcal{P}^e$ and $\rho \in \mathcal{O}^e$, then

(i) $Unl(E)\ must\ Unl(\rho)$ implies $E\ wfmust\ \rho$, but not the vice versa.

(ii) $Unl(E)\ must\ Unl(\rho)$ implies $E\ sfmust\ \rho$, but not the viceversa.

Proof. (Sketch of:) Consider item (i): the positive result is trivial, since a successful weak fair computation is obviously a successful maximal computation. To prove the negative result, consider $E := (\nu a)(!a_{v_1}.\bar{a}_{v_2} \mid \bar{a}_{v_3}) \mid \bar{b}_{v_4}$ and $\rho := b_{v_5}.\omega_{v_6}$. It is not difficult to check that $Unl(E)\ \not{must}\ Unl(\rho)$. $E\ wfmust\ \rho$ follows by the fact that,

given a weak fair computation from $E \mid \rho$, there has to exist a term performing ω, being v_5 already live since the beginning of the computation and having to lose its liveness at least once, by definition of weak fairness. In this case, losing liveness implies that b_{v_5} is performed. Item (ii) is a corollary of item (i) and Proposition 6.2. □

6.2 Weak and strong fairness vs fair testing semantics

Since weak fair must is strictly finer than strong one, the latter would look suitable to express fair testing semantics. However, Theorem 6.4 shows that not only the former but also the latter does not suffice to characterize fair testing semantics.

Theorem 6.4 For every $E \in \mathcal{P}^e$ and $\rho \in \mathcal{O}^e$, then

(i) E *wfmust* ρ implies $Unl(E)$ *fair* $Unl(\rho)$, but not the vice versa.

(ii) E *sfmust* ρ implies $Unl(E)$ *fair* $Unl(\rho)$, but not the vice versa.

Proof. (Sketch of:) Consider item (ii). Regarding the positive result, it is crucial to show that, given $S, S' \in \mathcal{E}^e$ such that $S' \stackrel{\varepsilon}{\Longrightarrow} S$, and a strong fair computation $\mathcal{C}$ from S, then the computation obtained by prefixing $\mathcal{C}$ with $S' \stackrel{\varepsilon}{\Longrightarrow} S$ keeps on being strong fair.

To prove the negative result of item (ii), it is enough to consider

$$E := \bar{c}_{v_1} \mid !c_{v_2}.(\nu a)(\bar{a}_{v_3} \mid a_{v_4}.\bar{c}_{v_5} \mid a_{v_6}.\bar{b}_{v_7})$$

and $\rho := b_{v_8}.\omega_{v_9}$. It easy to check that $Unl(E)$ *fair* $Unl(\rho)$, but there exists a strong fair computation in which v_8 never becomes live. Since v_8 prefixing ω_{v_9} is never performed and ω_{v_9} is the only ω occurrence along the given computation, the success will never be reached. Item (i) is just a corollary of item (ii). □

7 Strong fairness and fair testing semantics

To pick up the intuition behind the negative results in Theorem 6.4, we need a deeper explanation on what *live* means in the notion of strong (and weak) fairness. An action corresponding to a live label is not required to be performed to lose its liveness. Of course, when such an action is performed, then its label disappears forever. However, the label of an action may be present but no longer be live if, for example, a complementary action, which determines its liveness, is consumed in a synchronization with another partner.

We start explaining the implication from strong fair must to fair testing semantics: P *fair* o means that every maximal computation from $P \mid o$ potentially can always be successful i.e., from every state, action ω can be performed after finitely many interactions of live actions. The existence of a computation from $P \mid o$, where at least a state cannot lead to success at all (it means that P $\not$*fair* o), implies that from that state it will be impossible to reach ω for any fair scheduling of its actions.

However, there are experiments such that P *fair* o and the set of their maximal computations includes maximal unsuccessful computations. Consider, for instance, $P := \bar{c} \mid !c.(\nu a)(\bar{a} \mid a.\bar{c} \mid a.\bar{b})$ and $o := b.\omega$. Denote $Q_2 := (\nu a)(\bar{a} \mid a.\bar{c} \mid a.\bar{b})$. In the following (infinite) unsuccessful computation

$P \mid o = \bar{c} \mid !c.Q_2 \mid b.\omega \xrightarrow{\tau} !c.Q_2 \mid Q_2 \mid b.\omega \xrightarrow{\tau} \bar{c} \mid !c.Q_2 \mid (\nu a)(a.\bar{b}) \mid b.\omega \xrightarrow{\tau} ...$
$... \xrightarrow{\tau} \bar{c} \mid !c.Q_2 \mid (\nu a)(a.\bar{b}) \mid .. \mid (\nu a)(a.\bar{b}) \mid b.\omega \xrightarrow{\tau} ...$

ω is always prefixed and its prefix will never be performed, since any occurrence of $\bar{b}$ is prefixed in a deadlock term $(\nu a)(a.\bar{b})$. Notice that this computation is strong fair and unsuccessful. The prefix b in $b.\omega$ is not performed because it is always disabled, and this is allowed in the strong fairness definition, even if it could become live and be performed, in fair testing. Along a computation, strong fairness gives to live actions only a finite number of chances to be performed or be disabled; after finitely many steps, any action is either performed or disabled. The following result emphasizes the reason behind the impossibility of characterizing fair testing by strong (and weak) fairness.

Theorem 7.1 It is not possible to characterize *sfmust* and *wfmust* in terms of a fair testing-like semantics on the basis of the transition tree only.

Proof. Without loss of generality, we consider processes of the form $!P$, for a generic $P \in \mathcal{P}$. Consider the following processes: $P := (\tau.\bar{a}) \mid (\nu c)(!c.\bar{c} \mid \bar{c})$ and $Q := (!(\nu b)(\bar{b}|b|b.\bar{a}))|(\nu c)(!c.\bar{c}|\bar{c})$. Fairness assumptions distinguish P and Q: in fact, every strong (and weak) fair computation from P forces the execution of $\bar{a}$, sooner or later. This is not the case of some strong (and weak) fair computations from Q: occurrences of b and $b.\bar{a}$ compete to be performed infinitely often and, whenever one occurrence of b in $(\nu b)(\bar{b}|b|b.\bar{a})$ is performed, $b.\bar{a}$ is disabled forever, i.e. the fairness constraint has not effect anymore. It follows that P and Q are neither *sfmust* nor *wfmust* equivalent, i.e. there exists some observer o that distinguishes P and Q w.r.t. both *sfmust* and *wfmust*. However, if we only consider transitions out of the terms P and Q, they are even strong bisimilar. Hence, it follows that $(P \mid o)$ e $(Q \mid o)$ are strong bisimilar too, for every observer o. We can conclude that a fair testing definition would not distinguish P and Q. □

8 Related work

Fairness is a key concept in systems modeling and verification. Different kinds of fairness have been proposed in process algebras (see, for instance, [Hen87]). In this paper we adopt the definitions of weak and strong fairness proposed for CCS-like languages by Costa and Stirling in [CS84,CS87], to the π-calculus. An important result stated in [CS84,CS87] characterizes fair computations as the concatenation of certain finite sequences, called LP-steps that permits to think of fairness in terms of a 'localizable property' and not as a property of complete maximal executions. Almost simultaneously, two groups of authors [NC95], [BRV95] have come up with the so-called *fair testing*. They proposed two equivalent testing semantics with the property of abstracting from 'certain' divergences in contrast to the classical must testing. The idea is to modify the classical definition of must testing in such a way that the success can always be reached after finitely many steps. Both groups of authors present alternative characterizations of the new fair testing semantics. In [BRV96], the framework described in [BRV95] is extended to consider a set of sound axioms for fair testing and with more examples showing the usefulness of the new semantics. Another interesting paper is [FG98], where the authors generate

a natural hierarchy of equivalences for asynchronous name-passing process calculi based on variations of Milner and Sangiorgi's weak barbed bisimulation. The considered calculi (based on π-calculus and join calculus) are asynchronous in the sense of [HT91]. After defining a particular class of contexts, called *evaluation contexts* - contexts with only one hole and unguarded - they prove that barbed congruence coincides with Honda and Yoshida's reduction equivalence and, when the calculus includes name matching, with asynchronous labeled bisimulation. They also show that barbed congruence is coarser than reduction equivalence when only one barb is tested. By combining simulation coupling and barbed properties, they prove that every coupled barbed equivalence strictly implies fair testing equivalence. They show that both relations coincide in the join calculus and on a restricted version of the π-calculus where reception occurs only on names bound by a restriction (not on free names and not on received names). In [Koo85], Koomen explains fairness with probabilistic arguments: Fair Abstraction Rule says that no matter how small the probability of success, if you try often enough you will eventually succeed. The probabilistic intuitions motivating this rule are formalized in [NR99], where the authors define a probabilistic testing semantics which can be used to alternatively characterize *fair testing*. The key idea is to define this new semantics in such a way that two non-probabilistic processes are fair-equivalent if and only if any probabilistic version of both processes are equivalent in the probabilistic testing semantics. In order to get this result, the authors define a simple probabilistic must semantics, by saying that a probabilistic process *must* satisfy a test if and only if the probability with which the process satisfies the test equals 1. The subject of fairness in probabilistic systems has been widely discussed in the literature; Pnueli [Pnu83] introduces the notion of *extreme fairness* and *α-fairness*, to abstract from the precise values of probabilities.

9 Conclusion and future work

In this paper, we define a labeled version of the π-calculus [MPW92], importing techniques in [CS84,CS87] for CCS-like languages. We compare weak and strong fairness and prove that both notions of fairness are not enough to characterize fair testing semantics and we state the main reason of this failure. The results scale to the asynchronous π-calculus [Bou92] and do not depend on the proposed labeling method. As a future work, we plan to investigate on the existence of alternative characterizations of the investigated fairness notions, allowing simple and finite representations of fair computations such as the use of regular expressions as in [CDV03,CDV04]. It is also interesting to investigate on the impact that these different notions of fairness have on the encodings from the π-calculus into the asynchronous π-calculus [CCP05].

References

[BD95] M. Boreale, R. De Nicola, *Testing Equivalence for Mobile Processes*, Information and Computation, **120**, pp. 279-303, 1995.

[Bou92] G. Boudol, *Asynchrony and the π-calculus*, Technical Report 1702, INRIA, Sophia-Antipolis, 1992.

[BRV95] E. Brinksma, A. Rensink, W. Vogler, *Fair Testing*, Proc. of CONCUR'95, LNCS, **962**, pp. 313-327, 1995.

[BRV96] E. Brinksma, A. Rensink, W. Vogler, *Applications of Fair Testing*, In "Protocols Specification, Testing and Verification" (XVI), Chapman & Hall, pp. 145-160, 1996.

[CCP05] D.R. Cacciagrano, F. Corradini, C. Palamidessi, *Separation of synchronous and Asynchronous Communication via Testing*. In 12th International Workshop on Expressiveness in Concurrency, EXPRESS'05, 2005.

[CDV03] F.Corradini, M.R.Di Berardini, W.Vogler, *Relating Fairness and Timing in Process Algebra*, Proc. of Concur'03, LNCS, **2761**, pp. 446-460, 2003.

[CDV04] F.Corradini, M.R.Di Berardini, W.Vogler, *Fairness of Components in System Computations*. In 11th International Workshop on Expressiveness in Concurrency, EXPRESS'04, 2004.

[FG98] C. Fournet, G. Gonthier, *A Hierarchy of Equivalences for Asynchronous Calculi*, Proc. of ICALP'98, pp. 844-855, 1998.

[CS84] G. Costa, C. Stirling, *A Fair Calculus of Communicating Systems*, Acta Informatica, **21**, pp. 417-441, 1984.

[CS87] G. Costa, C. Stirling: *Weak and Strong Fairness in CCS*. Information and Computation **73**, pp. 207-244, 1987.

[DH84] R. De Nicola, M. Hennessy, *Testing Equivalence for Processes*, Theoretical Computers Science, **34**, pp. 83-133, 1984.

[Fra86] N. Francez, *Fairness*, Springer-Verlag, 1986.

[Hen87] M. Hennessy, *An Algebraic Theory of Fair Asynchronous Communicating Processes*, Theoretical Computer Science, **49**, pp. 121-143, 1987.

[HT91] K. Honda, M. Tokoro. *An Object calculus for Asynchronous Communication*, Proc. of ECOOP '91, LNCS, **512**, pp. 133-147, 1991.

[HY94] K.Honda, N.Yoshida, *Replication in Concurrent Combinators*, Proc. of TACS '94, LNCS, **789**, 1994.

[Koo85] C. Koomen, *Albegraic Specification and Verification of Communications protocols*, Science of Computer Programming, **5** pp. 1-36, 1985.

[LPS81] D. Lehmann, A. Pnueli, J. Stavi, *Impartiality, justice and Fairness:the Ethics of Concurrent Termination*, Proc. of 8th Int. Colloq. Aut. Lang. Prog., LNCS, **115**, pp. 264-277, 1981.

[Mil89] R. Milner, *Communication and Concurrency*, Prentice-Hall International, 1989.

[MPW92] R. Milner, J. Parrow, D. Walker, *A Calculus of Mobile Processes*, Part I and II, Information and Computation, **100**, pp. 1-78, 1992.

[NC95] V. Natarajan, R. Cleaveland, *Divergence and Fair Testing*, Proc. of ICALP'95, LNCS, **944**, pp. 648-659, 1995.

[NR99] M. Núñez, D. Rupérez, *Fair testing through probabilistic testing*, Acta Informatica, **19**, pp. 195-210, 1983. Protocol Specification, Testing, and Verication, **19**, Kluwer Academic Publishers, pp. 135-150, 1999.

[Pnu83] A. Pnueli, *On the Extremely Fair Treatment of Probabilistic Algorithms*, Proc. of ACM Symph. Theory of Comp., pp. 278-290, 1983.

[QS83] J.P.Queille, J.Sifakis, *Fairness and Related Properties in Transition Systems-A Temporal Logic to Deal with Fairness*, Acta Informatica, **19**, pp. 195-210, 1983.

Compositional state space reduction using *untangled* actions

Xu Wang Marta Kwiatkowska[1]

School of Computer Science, University of Birmingham
Edgbaston, Birmingham B15 2TT, UK

Abstract

We propose a compositional technique for efficient verification of networks of parallel processes. It is based on an automatic analysis of LTSs of individual processes (using a failure-based equivalence which preserves divergences) that determines their sets of "conflict-free" actions, called untangled actions. Untangled actions are compositional, i.e. synchronisation on untangled actions will not destroy their "conflict-freedom". For networks of processes, using global untangled actions derived from local ones, efficient reduction algorithms can be devised for systems with a large number of small processes running in parallel.

Keywords: Untangled action, Conflict-freedom, Partial order reduction, Process algebra, Compositionality, Determinism, and Partial confluence.

1 Introduction

Informally, an untangled action [2] is a special action in a discrete event system of causality and conflict [24]. At any state of the system the action, if enabled, shall not be entangled through any conflict with the rest of the system, and its only contribution to the system dynamics is by causality. Therefore, if an untangled action is not observed (due to hiding or other operations), its occurrence becomes time irrelevant [3]. This gives us the opportunity to reduce the search space by considering only one possibility of its occurrence time.

The notion of untangled actions is closely related to similar ideas in true concurrency semantics [24], partial order reduction [12,21], and Petri net unfolding [10]. Within process algebra, the closest work to ours is that of τ-confluence reduction by Groote, van de Pol, Blom etc [8,7,2,3].

[1] {X.Wang,M.Z.Kwiatkowska}@cs.bham.ac.uk

[2] We prefer to use here the term "action" instead of "event" so as to distinguish between actions and their occurrences. But in the rest of the paper they may be used interchangeably.

[3] Some type of progress/maximality assumption is needed to guarantee that the action will eventually occur.

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

2 Motivations

This work is motivated by our experience in using process algebra (e.g. CSP and FDR2 [6]) to verify asynchronous circuits [22], where high concurrency in gate-level circuits induces serious state explosion problems. A well-known example is the tree arbiter [5,25]. A tree arbiter consists of a tree of arbiter cells. Each arbiter cell behaves as a two-way arbiter for its sons while at the same time acting as one of the clients of its father node. In this way, a tree arbiter implements multi-way arbitration through a hierarchy of two-way arbitration.

The state space of tree arbiters blows up exponentially wrt. the tree size, and it is not readily amenable to reduction due to the conflicts inherent to arbitration. Previously, Petri net unfolding techniques [10] and partial-order reduction enhanced BDD methods [1] had been applied to it, with limited success. In this paper, we will propose untangled action as a viable solution to this and similar systems.

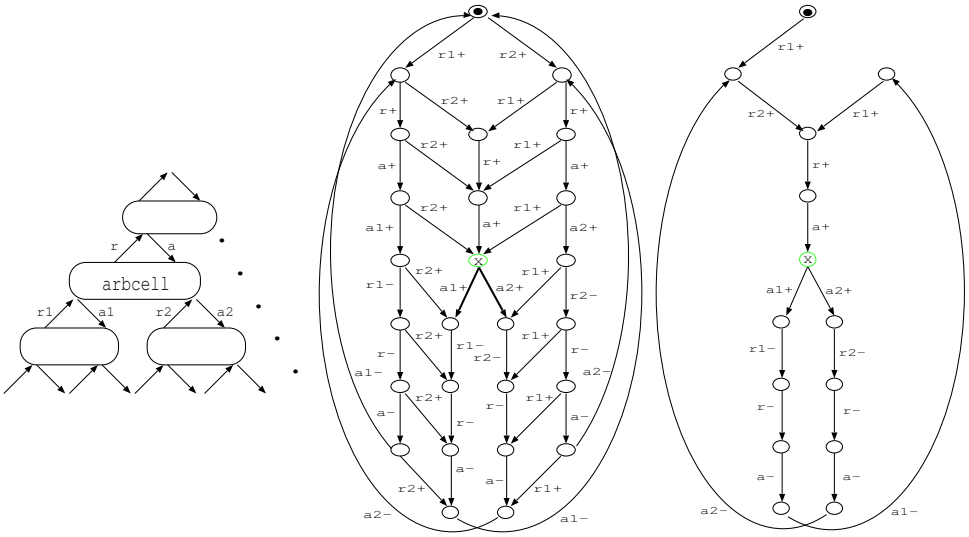


Figure 1. An arbiter tree, and the original and reduced LTSs of an arbiter cell

Untangledness is a simple idea. We will defer the theoretical justification to later sections. For the arbiter cell example, it is not difficult to see that only two actions are entangled in conflicts, i.e. $a1+$ and $a2+$. These tangled actions coincide with so called *output choice* signal transitions [4]. With this information, it is straightforward to give a state-space reduction algorithm by prioritising untangled actions (similar to the *chase* reduction in FDR2) in the exploration of the state space. That is, in a depth-first search, given a state with a non-empty set of untangled outgoing transitions, we use some *strategy* to pick and prioritise one from the set to explore; all the other transitions from the same state, untangled or tangled, will be completely ignored in the exploration. In case a state has no untangled transition, all the transitions from that state will need to be explored. It is not difficult to apply the algorithm to reduce the LTS of the arbiter cell manually. Figure 1 gives the reduced state space based on one possible prioritisation strategy.

However, the above reduction is correct only if we treat the arbiter cell as a closed system and all the untangled actions are not observable to the checked properties. Synchronisation with the environment may introduce new conflicts that can destroy

the untangledness of the actions. The previous works on τ-confluence solve these problems by considering only τ action [7,2,3] or *locally visible and globally invisible* (lvgi) actions [4] without synchronisation [11].

Secondly, untangledness analysis with the environment factors taken into account is difficult since the analysis must avoid explicitly working on the global state space, which is intractable in our context. In τ-confluence reduction, the proposed solution is to use theorem proving on a symbolic representation (so called *linear processes*) of global state spaces [3], or to use compositionality [11] as we have adopted. However, since the involved actions must be synchronisation-free, their compositionality does not apply to the tree arbiter, whose lvgi actions, e.g. r, a, $r1$, etc., need further synchronisation.

In this paper, we propose a compositional technique for concurrent systems such that untangledness analysis is done at a local level. A compositionality theorem automatically calculates global untangled actions from the local ones. Using thus obtained results, reduction can be applied on-the-fly on the global systems.

Structure of the paper. After the introduction of basic notations (Section 3) and concurrent systems (Section 4), two important (partial) determinacy notions on LTSs with lvgi actions, one stronger than the other, are proposed in Section 5. The former is compositional on the lvgi actions without synchronisation potential and induces a simple and efficient on-the-fly reduction procedure (Section 6). The latter removes the synchronisation restriction and becomes compositional on all lvgi actions, and thus enables compositional reductions (Section 7). Preliminary experiment results are given and the paper is summarised in Section 8.

3 Basic Notation

A *LTS* (*Labelled Transition System*) is (A, S, T, s^0), consisting of a set A of visible events called the alphabet, a (finite or infinite) set S of states, a transition relation $T : S \times (A \cup \{\tau\}) \leftrightarrow S$ and the initial state $s^0 \in S$.

(**Event, Sequence and Trace**) Let e be a visible event, a be a τ or visible event, and Δ be a subset of A. Δ^τ and A^τ denote $\Delta \cup \{\tau\}$ and $A \cup \{\tau\}$. k and l are finite sequences of events (including the empty sequence, ϵ) [5]. t and u are finite traces, i.e. finite sequences of non-τ events. $\overline{k}$, $\overline{l}$, $\overline{t}$ and $\overline{u}$ are the infinite variants, while $\tilde{k}$, $\tilde{l}$, $\tilde{t}$ and $\tilde{u}$ can denote both finite and infinite ones.

(**Sequence operations**) Juxtaposition is used for sequence concatenation, e.g. $k\tilde{l}$. $|\tilde{k}|$ gives us the length of the sequence $\tilde{k}$ (ω for infinity). *head* and *tail* (unary prefix operator) are defined as normal. $-$ is a binary infix operator removing from a sequence left to right all the members in another sequence according to multiplicity, e.g. $e_1 e_2 e_2 e_3 e_1 e_2 - e_3 e_2 e_2 = e_1 e_1 e_2$ and $e_1 e_2 - e_2 e_3 = e_1$.

(**Prefix order, Projection and Containment**) $\leq$ is the prefix order on sequences. $pref()$ calculates the set of (finite) prefixes on a sequence. $\tilde{k} \upharpoonright_\Delta$ removes from $\tilde{k}$ all the events not in Δ. They both can be lifted to operate on sets of sequences. We say $\tilde{l}$ *contains* $\tilde{k}$ iff $\forall\, a \in A^\tau \bullet \tilde{k} \upharpoonright_{\{a\}} \leq \tilde{l} \upharpoonright_{\{a\}}$; $\tilde{l}$ *trace-contains* $\tilde{k}$ iff $\tilde{l} \upharpoonright_A$

[4] Formally, given a network of processes, a lvgi action is one that is visible on an individual process but is eventually hidden during process composition and thus invisible at the global network level.

[5] Sometimes, a and e are also used as singleton sequences or traces.

contains $\tilde{k} \restriction_A$.

Definition 3.1 [Path and Arrow notation] Given a LTS, (A, S, T, s^0):

- a finite path is a finite sequence of alternating states and events, $s_0 a_1 s_1 a_2 ... a_n s_n$ ($\in PATH \mathrel{\widehat{=}} S \times (A^\tau \times S)^*$), where $(s_{i-1}, a_i, s_i) \in T$ for all $1 \leq i \leq n$. The labelling sequence of the path is $a_1 a_2 ... a_n$ (i.e. ϵ when $n = 0$). Similarly, $s_0 a_1 s_1 a_2 ...$ ($\in \overline{PATH} \mathrel{\widehat{=}} S \times (A^\tau \times S)^\omega$) is an infinite path and $a_1 a_2 ...$ is its labelling sequence.
- $s \xrightarrow{k} s'$ iff there is a k-labelled finite path going from s to s'. $s \xrightarrow{\overline{k}}$ iff there is a $\overline{k}$-labelled infinite path starting from s.
- $\xrightarrow{a} s'$ iff there exists a reachable state s in the LTS such that $s \xrightarrow{a} s'$, and we say s' is *caused* by a.
- $s \stackrel{t}{\Longrightarrow} s'$ iff $s \xrightarrow{k} s'$ and $k \restriction_A = t$; $s \stackrel{\overline{t}}{\Longrightarrow}$ iff $s \xrightarrow{\overline{k}}$ and $\overline{k} \restriction_A = \overline{t}$; $s \stackrel{t}{\Longrightarrow}$ iff there exists a state s' such that $s \stackrel{t}{\Longrightarrow} s'$.

A state s is *deadlocked*, i.e. $deadlock(s)$, iff s does not have any outgoing transition. A state s is *divergent*, i.e. $divergent(s)$, iff there is an infinite τ-path in the LTS that starts from s.

Definition 3.2 [Traces] Given a LTS, the set of finite traces FT is $\{t \mid s^0 \stackrel{t}{\Longrightarrow}\}$, the set of infinite traces IT is $\{\overline{t} \mid s^0 \stackrel{\overline{t}}{\Longrightarrow}\}$, the set of deadlock traces LT is $\{t \mid \exists\, s \bullet deadlock(s) \wedge s^0 \stackrel{t}{\Longrightarrow} s\}$, and the set of divergence traces DT is $\{t \mid \exists\, s \bullet divergent(s) \wedge s^0 \stackrel{t}{\Longrightarrow} s\}$.

Definition 3.3 [Normalisation] A LTS, LTS^N, is *normalised* iff all τ-transitions are self-loops, i.e. $s \xrightarrow{\tau} s' \Rightarrow s = s'$, and there is no ambiguous transition, i.e. $s \xrightarrow{e} s' \wedge s \xrightarrow{e} s'' \Rightarrow s' = s''$.

4 Concurrent Systems

Basic processes given as LTSs can be combined using the parallel and the hiding operators to form a concurrent system [15].

$$SCS ::= LTS \mid SCS \parallel SCS' \mid SCS \setminus \Delta \mid SCS[R^{1-1}]$$

Definition 4.1 [Parallel] Given LTS_1 and LTS_2, $LTS_1 \parallel LTS_2$ gives another LTS, (A, S, T, s^0), where $A = A_1 \cup A_2$, $S = S_1 \times S_2$, $s^0 = (s_1^0, s_2^0)$ and T is the least relation satisfying the following rules:

$$\frac{s_1 \xrightarrow{a} s_1' \quad a \notin A_2}{(s_1,s_2) \xrightarrow{a} (s_1',s_2)} \qquad \frac{s_2 \xrightarrow{a} s_2' \quad a \notin A_1}{(s_1,s_2) \xrightarrow{a} (s_1,s_2')} \qquad \frac{s_1 \xrightarrow{e} s_1' \quad s_2 \xrightarrow{e} s_2'}{(s_1,s_2) \xrightarrow{e} (s_1's_2')}$$

Definition 4.2 [Hiding] Given LTS, $LTS \setminus \Delta$ gives a new LTS, (A', S, T', s^0), where $A' = A \setminus \Delta$, and $T' = T'[\tau/\Delta]$, i.e. substituting τ for every Δ event on every occurrence in T.

Definition 4.3 [Renaming] Given LTS, $LTS[R]$, where $R : A \leftrightarrow A'$ and $\mathrm{dom}\, R \cap \mathrm{ran}\, R = \{\}$, gives a new LTS, (A', S, T', s^0), where $A' = (A \setminus \mathrm{dom}\, R) \cup \mathrm{ran}\, R$, and $T' = \{(s, a, s') \mid (a \notin \mathrm{dom}\, R \wedge (s, a, s') \in T) \vee (\exists\, e \bullet (e, a) \in R \wedge (s, e, s') \in T)\}$.

The definition allows m-to-n renaming. Usually only special cases are needed: 1-to-1 (R^{1-1}), 1-to-m (R^{1-m}) and m-to-1 (R^{m-1}) [6].

4.1 Semantics

In classic CSP [15], stable failures and failure/divergences are the major semantic models used in CSP. They are both finite trace models. However, there is a newly developed infinite trace CSP model [16], the $\mathcal{SBD}$ model, which preserves all the divergence traces in CSP processes.

Given a LTS, (A, S, T, s^0), a state s is *stable*, i.e. $stable(s)$, iff $\neg(s \xrightarrow{\tau})$. Sometimes, we also use $stable(s, \Delta)$ to mean $\forall\, a \in \Delta \bullet \neg(s \xrightarrow{a})$. Given a set of finite sequences, $lmt()$ outputs a set of infinite sequences, each being the limit of a chain of increasing (i.e. prefix order) finite sequences belonging to the set. Define $IB \mathrel{\widehat{=}} IT \cup lmt(DT)$.

(**Stable failures and Behaviours**) The set of stable traces ST is $\{t \mid \exists\, s \bullet stable(s) \wedge s^0 \stackrel{t}{\Longrightarrow} s\}$. The set of stable failures SF is $\{(t, \Delta) \mid \exists\, s \bullet stable(s) \wedge s^0 \stackrel{t}{\Longrightarrow} s \wedge \forall\, e \in \Delta \bullet \neg\, (s \xrightarrow{e})\}$. The type of *behaviours* is $BEHV(A) \mathrel{\widehat{=}} A^* \cup (A^* \cdot \{\tau^\omega\}) \cup A^\omega$. The set of behaviours BH is $\{\tilde{k} \mid \tilde{k} \in FT \vee \tilde{k} \in IT \vee (\tilde{k} = t\tau^\omega \wedge t \in DT)\}$.

Definition 4.4 [SBD equivalences] $LTS \stackrel{SBD}{=} LTS'$ iff $SBD(LTS) = SBD(LTS')$, and $LTS \stackrel{SBDF}{=} LTS'$ iff $SBDF(LTS) = SBDF(LTS')$, where $SBD(LTS) = (FT, DT, IB)$ and $SBDF(LTS) = (SF, DT, IB)$.

Theorem 4.5 (Weakest congruence [20,13]) *W.r.t. the parallel, hiding and renaming operators defined above* [7]*, $\stackrel{SBDF}{=}$ is the weakest congruence preserving LT and DT information on LTSs.*

Definition 4.6 [U-determinism] Given a LTS, it is *U-deterministic* (i.e. unstably deterministic [8]) iff $ST \cap DT = \{\}$ and $te \in FT \Rightarrow (t, \{e\}) \notin SF$.

Proposition 4.7 *Given U-deterministic LTS and LTS′, $LTS \stackrel{SBDF}{=} LTS'$ iff $LTS \stackrel{SBD}{=} LTS'$. Given U-deterministic LTS, there exists a normalised LTS^N such that $LTS \stackrel{SBDF}{=} LTS^N$.*

Proposition 4.8 *U-deterministic LTSs are closed under the parallel composition.*

---

[6] The m-to-n renaming and interface parallel operators can be reduced to the 1-1 renaming and alphabetised parallel operators using transformations.

[7] The weakest congruence result can be extended to the other CSP operators [15] since $SBDF$ is a congruence on those CSP operators as well [16].

[8] As suggested by Roscoe [19], U-determinism does not quite coincide with the operational intuition of determinism, e.g. it does not possess τ-inertness [8]. In this sense, detachability on an empty A_i set (Section 5) is closer to this operational intuition. A thorough investigation of operational determinism has recently been done by Hansen and Valmari [9]

5 Untangled action analysis

Given a LTS consisting of τ action, lvgi actions and globally visible actions, the most important ingredient of its state space traversal algorithm probably is, at a state with multiple outgoing transitions, how to choose the next branch to pursue. The decision can be split into two parts. One is what we call *visible choices*, which decide the next visible action in the global behaviour. The other is called *invisible choices*; they decide which specific next branch to follow in order to achieve the objective of the visible choice. τ and lvgi transitions, as well as ambiguous transitions, give rise to invisible choices. Usually, visible choices are intertwined with invisible choices. But under certain conditions, they can be separated or *detached* from each other in the sense that they are independent to each other. That is, no matter what invisible choice is taken, it will not affect the achievement of the decided visible choice. For the case that ambiguous transitions are not considered, this is τ-*inertness* [8].

This insight leads to the state space reduction algorithms in many process algebraic frameworks [14,11,2,6]. The algorithm simply makes arbitrary decision on invisible choices and ignores the other alternatives completely in the state space traversal. It also forms the basis of our reduction algorithm in Section 6. We call such systems *detachable* systems.

5.1 Detachability

A LTS is regarded as an acceptor of behaviours. Its alphabet A is partitioned by A_v and A_i, which are respectively the set of globally visible actions and the set of lvgi actions. When being fed a global behaviour (i.e. $\tilde{k} \in BEHV(A_v)$), the acceptor will control the invisible choices in the LTS and try to accept or reject the behaviour. The different ways an acceptor decides on invisible choices give rise to different acceptor strategies. Thus a strategy can be regarded as an unfolding of the original LTS followed by a reduction that resolves all the invisible choices in it, e.g. the reduced LTS in Figure 1 is a strategy of the original one. For the same behaviour, the acceptor may have both a strategy to accept it and one to reject it.

Given LTS, formally a strategy, $stg : PATH \times BEHV(A_v) \nrightarrow (A^\tau \times S) \cup \{stop, reject\}$, is a minimal (subset order) partial function satisfying the rules:

(i) $\{s^0\} \times BEHV(A_v) \subseteq \mathrm{dom}\, stg$

(ii) $(s_0 a_1 ... s_n, e\tilde{k}) \in \mathrm{dom}\, stg \Rightarrow stg(s_0 a_1 ... s_n, e\tilde{k}) \in \{(a, s) \mid s_n \xrightarrow{a} s \wedge a \in A_i^\tau \cup \{e\}\} \cup \{reject \mid \neg\, s_n \xrightarrow{e} \wedge\, stable(s_n, A_i^\tau)\}$

(iii) $(s_0 a_1 ... s_n, \epsilon) \in \mathrm{dom}\, stg \Rightarrow stg(s_0 a_1 ... s_n, \epsilon) \in \{(a, s) \mid s_n \xrightarrow{a} s \wedge a \in A_i^\tau\} \cup \{stop \mid stable(s_n, A_i^\tau)\}$

(iv) $(s_0 a_1 ... s_n, \tau^\omega) \in \mathrm{dom}\, stg \Rightarrow stg(s_0 a_1 ... s_n, \tau^\omega) \in \{(a, s) \mid s_n \xrightarrow{a} s \wedge a \in A_i^\tau\} \cup \{reject \mid stable(s_n, A_i^\tau)\}$

(v) $stg(s_0 a_1 ... s_n, \tilde{k}) = (a, s) \Rightarrow (s_0 a_1 ... s_n as, \tilde{k} - (a \restriction_{A_v})) \in \mathrm{dom}\, stg$

Initially, the acceptor is ready to be fed with any behaviour (**rule 1**). Once fed, the acceptor starts the execution to consume the sequence step by step (**rule 5**). A state of the execution (i.e. the input to the function) consists of a *history* (a finite path in LTS whose labelling sequence, after the projection onto A_v, gives a prefix of the

fed behaviour denoting the consumed part) and a *suffix* of the behaviour (denoting the remaining part). Minimality of the function implies that only reachable states are defined on *stg*. Given a reachable state and a *pending* action, i.e. e on top of the current suffix, the acceptor is free to make any invisible choice to transit in *LTS*, e.g. (a, s), so long as the transition is consistent with e (the visible choice), i.e. $a \in A_i^\tau \cup \{e\}$ (**rule 2**). When the execution reaches a state where no more consistent invisible choice can be made, the acceptor will either stop (if the consumption is complete) or reject (if incomplete). Given a behaviour $\tilde{k}$ and a strategy *stg*, the acceptor's execution produces a finite path if it ends with *stop* or *reject*; otherwise the execution produces an infinite path.
(**Acceptance condition**) We say *stg* is an *accepting* strategy for $\tilde{k}$ on *LTS* iff the execution does not end with *reject* and produces a path whose labelling sequence trace-containing $\tilde{k}$. Otherwise, *stg* is a *rejecting* strategy for $\tilde{k}$ on *LTS*.

However, these strategies do not handle divergence correctly. For instance, if the initial state of *LTS* has a τ loop, *LTS* has a simple rejecting strategy (i.e. following the τ loop indefinitely) for any non-trivial A_v behaviours. It is "unfair" for systems like normalised LTSs, where the τ loops are self-loops, causing no state change ("unprogressing loops"). Thus, an extra requirement shall be put on strategies.

Definition 5.1 [Fairness] A strategy *stg* is a *fair* strategy iff its (infinite) execution cannot lead to a state after which, though an action e is pending to be consumed (c.f. **rule 2**), it makes no further A_v transition and an action $a \in A_i^\tau \cup \{e\}$ is always enabled (i.e. on *LTS*) but never taken.

The definition is a kind of maximality and weak fairness requirement on actions as that in partial order semantics. However, it is not applied on all actions. Only the pending e and the A_i^τ actions will be guaranteed progress. Progress on e will be able to guide strategies out of unprogressing loops. Progress on the A_i^τ actions can guide strategies out of indefinite delays on any member of A_i^τ. It is necessary for compositionality.

Definition 5.2 [May&Must acceptance] A LTS *may-accept* a behaviour iff there exists an accepting strategy. It *must-accept* a behaviour iff there does not exist a fair rejecting strategy.

It is not difficult to see that the set of may-accepted behaviours is exactly $BH(LTS \setminus A_i)$ and thus implies the *SBD*-equivalence.

Definition 5.3 [Detachability] Given *LTS* and $A_i \cup A_v = A$, A_i is detachable from *LTS* (or *LTS* is detachable on A_i) iff *LTS* may-accept $\tilde{k}$ iff *LTS* must-accept $\tilde{k}$ for all $\tilde{k} \in BEHV(A_v)$.

Detachability has many good properties, some of which will be shown in Section 6, where a reduction algorithm based on detachability will be given. Here we will just mention U-determinism and a restricted form of compositionality.

Proposition 5.4 *Δ is detachable from LTS implies LTS $\setminus$ Δ is U-deterministic, but not vice versa.*

It is *crucial*, however, to notice that detachable *LTS* on Δ does not imply

detachable $LTS \setminus \Delta$ (i.e. on $\{\}$). Hiding removes the distinctiveness among the members of Δ; thus less progress requirement is placed on strategies and the fair rejection of behaviours becomes easier. Indeed, Hiding shall not be applied on LTSs any sooner than the reduction algorithm of Section 6 has used the distinctiveness.

Detachability is compositional (c.f. Theorem 7.1 for its exact formulation) if all lvgi actions concerned are synchronisation-free. It is in part thanks to the progress requirement on the A_i^τ actions. For example, $R = e \rightarrow R$ is detachable on $\{e\}$ and $R' = e' \rightarrow e'' \rightarrow Stop$ is detachable on $\{e'\}$ (i.e. even without any progress requirement). But, without the progress requirement on $\{e'\}$, $R \parallel R'$ is not detachable on $\{e, e'\}$.

On the other hand, compositionality does not hold for lvgi actions with synchronisation potential. Informally, it is due to the fact that detachability allows conflicts within A_i actions (an extreme case is "auto-conflict" within one action). It is just that the resolution of these conflicts does not affect the causality drive onto the A_v part which makes these conflicts detachable from those of the A_v part. Once there is synchronisation, conflicts can be propagated amongst processes and create new ones that may not be detachable.

The following two processes give an example:

$$P = e \rightarrow e \rightarrow e' \rightarrow Stop \;\Box\; e \rightarrow e' \rightarrow Stop$$
$$Q = e \rightarrow e' \rightarrow Stop$$

With $A_i = \{e\}$, P and Q are both detachable, although P contains an auto-conflict in the sense that one branch needs two e actions to enable e' while the other needs just one. The parallel composition of the two, however, is not detachable since one branch will lead to the occurrence of e' while the other will not. Therefore, to make compositionality fully work conflicts must be ruled out completely on A_i actions. This gives us the notion of *untangled actions*.

5.2 Untangledness

With synchronisation on lvgi actions, untangledness shall be sensitive to the type as well as the number of lvgi actions expended to drive causality.

Given LTS^N [9] and $A_i \cup A_v = A$, a strategy, $stg : PATH \times BEHV \nrightarrow (A^\tau \times S) \cup \{stop, reject\}$, is a minimal partial function satisfying:

(i) $\{s^0\} \times BEHV \subseteq \mathrm{dom}\, stg$

(ii) $(s_0 a_1 ... s_n, e\tilde{k}) \in \mathrm{dom}\, stg \Rightarrow stg(s_0 a_1 ... s_n, e\tilde{k}) \in \{(a, s) \mid s_n \xrightarrow{a} s \wedge a \in A_i^\tau \cup \{head(e\tilde{k} \upharpoonright_{A_v})\}\} \cup \{reject \mid \neg\, s_n \xrightarrow{e}\}$

(iii) $(s_0 a_1 ... s_n, \epsilon) \in \mathrm{dom}\, stg \Rightarrow stg(s_0 a_1 ... s_n, \epsilon) \in \{(a, s) \mid s_n \xrightarrow{a} s \wedge a \in A_i^\tau\} \cup \{stop \mid stable(s_n, A_i^\tau)\}$

(iv) $(s_0 a_1 ... s_n, \tau^\omega) \in \mathrm{dom}\, stg \Rightarrow stg(s_0 a_1 ... s_n, \tau^\omega) \in \{(a, s) \mid s_n \xrightarrow{a} s \wedge a \in A_i^\tau\} \cup \{reject \mid stable(s_n)\}$

(v) $stg(s_0 a_1 ... s_n, \tilde{k}) = (a, s) \Rightarrow (s_0 a_1 ... s_n as, \tilde{k} - (a \upharpoonright_A)) \in \mathrm{dom}\, stg$

[9] A definition based on unnormalised LTSs is also possible. But it complicates the presentation and the generality is not needed for this paper.

Like the previous one, the acceptor controls the order and the occurrence of A_i actions. Thus **rule 3** and the parts of **rule 2 and 4** not involving *reject* remain the same. Unlike the previous one, A_i actions become visible in the fed behaviours (**rule 1 and 5**) and the acceptor is more sensitive (the parts of **rule 2 and 4** involving *reject*). For instance, once the right type and number of actions have occurred (i.e. removed from the fed behaviours), a new action will be enabled on top of the current suffix (i.e. the pending e). e cannot be delayed by any other A action; if it is not simultaneously enabled on LTS, it may result in the immediate issue of *reject* (the *reject* part of **rule 2**). It gives the acceptor more freedom in rejecting behaviours (e.g. eee' behaviour of the P process above will incur *reject*). The acceptance conditions remain the same except for the adaptation for A_i visibility.
(**Acceptance condition**) *stg* is an *accepting* strategy for $\tilde{k}$ on LTS iff the execution does not end with *reject* and produces a path whose labelling sequence trace-containing $\tilde{k}$. Otherwise, *stg* is a rejecting strategy for $\tilde{k}$ on LTS.

Similarly, fairness can be simplified since the fed behaviours (with A_i visible) can guide itself now.

Definition 5.5 [Fairness] A strategy *stg* is a *fair* strategy iff its (infinite) execution cannot lead to a state after which the pending action e is always enabled (i.e. on LTS^N) but never taken.

Moreover, if we distinguish infinite rejecting strategies (i.e. infinite executions not trace-containing the fed behaviour) from finite ones (i.e. finite executions ending with *reject*), it is obvious that only finite rejecting strategies are needed.

Proposition 5.6 (Finite rejection) *If there is an infinite fair rejecting strategy for a behaviour, there is also a finite one for it.*

May-acceptance and must-acceptance can be defined like in the previous section. However, $BH(LTS^N)$ is only a subset of the may-accepted behaviours, and the definition of untangledness shall change accordingly.

Definition 5.7 [Untangledness] Given LTS^N (and $A_i = \Delta$), Δ is untangled in LTS^N (or LTS^N is untangled on Δ) iff LTS must-accept $\tilde{k}$ for all $\tilde{k} \in BH$.

Given Δ, its untangledness decision problem can be solved by a CSP refinement check using stable failures model in Appendix A. This is due to the finite rejection property. The LHS of the check (i.e. the specification) is a fixed process while the RHS is two copies of LTS^N coordinated by another fixed process. The refinement problem of this form is in NLOGSPACE.

Proposition 5.8 *Untangled action sets are closed under subset-hood and union.*

(**Maximal untangled set**) Given Δ, its maximal subset of untangled actions can be found by doing the CSP check on each singleton subset of Δ and taking the union of the successful ones.

Theorem 5.9 *Untangled Δ in LTS^N is also detachable (not vice versa).*

Note that untangledness, detachability, U-determinism etc. form a hierarchy of partial determinacy properties. An interesting discussion of various determinacy

and confluence notions in classic process algebras can be found in [18], where may-testing and must-testing are also used to characterise determinacy. Confluence in our context is the same as the untangledness on A (i.e. the full alphabet).

Untangled actions are compositional; global untangled actions can be calculated from local ones. The compositionality theorem will be given in Section 7, where a new compositional reduction technique enabled by it is also proposed. The new technique feeds the global untangledness information to a specially designed on-the-fly reduction procedure called $chase^+$, which reduces state spaces by exploiting detachability (c.f. Theorem 5.9).

6 Reduction Algorithm

The new algorithm is an extension of the *chase* function in FDR2 [6], and also shares similarity with the reduction algorithms based on τ-inertness [2,11,14]. The idea is based on the fact that in a detachable system a behaviour is accepted by its LTS iff it is accepted by a fair strategy of the LTS. Thus the LTS can be reduced by removing all other strategies in it, which results in an equivalent LTS containing just one strategy. The most important ingredient of the reduction algorithm, consequently, is finding the suitable fair strategy.

(**Round robin strategy**) Assume the actions in A_i^τ are arranged in a (directed) cycle with a default starting position, and $next(c, \Delta)$ is a function, which, given the current action c and the set of candidate actions Δ, outputs the candidate following c the closest in the cycle. (Note that, when $c = \epsilon$, the default starting position is assumed.) A subclass of fair strategies on finite state LTSs, called *round robin* strategies, use a round robin strategy on the cycle to implement fairness. Formally they are minimal partial functions satisfying the same conditions as in Section 5.1 but with **rule 2** replaced by the following [10]:

$2a'.$ $(s_0 a_1 ... a_n s_n, e\tilde{k}) \in \mathrm{dom}\, stg \wedge \neg\, fair\_loop(s_0 a_1 ... a_n s_n) \Rightarrow$
$stg(s_0 a_1 ... a_n s_n, e\tilde{k}) \in \{(a, s) \mid s_n \xrightarrow{a} s \wedge a = next(a_n \restriction_{A_i^\tau}, \{a : A_i^\tau \mid s_n \xrightarrow{a}\})\} \cup \{(e, s) \mid s_n \xrightarrow{e} s \wedge stable(s_n, A_i^\tau)\} \cup \{reject \mid \neg\, s_n \xrightarrow{e} \wedge\, stable(s_n, A_i^\tau)\}$

$2b'.$ $(s_0 a_1 ... a_n s_n, e\tilde{k}) \in \mathrm{dom}\, stg \wedge fair\_loop(s_0 a_1 ... a_n s_n) \Rightarrow$
$stg(s_0 a_1 ... s_n, e\tilde{k}) \in \{(e, s) \mid s_n \xrightarrow{e} s\} \cup \{(a, s) \mid \neg\, s_n \xrightarrow{e} \wedge\, s_n \xrightarrow{a} s \wedge a \in A_i^\tau\} \cup \{reject \mid \neg\, s_n \xrightarrow{e} \wedge\, stable(s_n, A_i^\tau)\}$

where $fair\_loop(s_0 a_1 ... s_n)$ is true iff the maximal suffix of $s_0 a_1 ... s_n$ that is a A_i^τ-path, say $s_i a_{i+1} ... s_n$, contains a fair A_i^τ-loop but $s_i a_{i+1} ... s_{n-1}$ does not. A fair A_i^τ-loop is a A_i^τ-loop that has gone through at least one round of the cycle.

Intuitively, this means that the strategy will give priority to A_i^τ transitions as long as the A_i^τ transitions on top of the history have not formed a fair A_i^τ loop yet. Once one is formed (and exactly at this moment) the pending e transition will be given priority (to implement the weak fairness on e). Thereafter, A_i^τ transitions continue to have priority.

Proposition 6.1 *Given LTS and A_i, a round robin strategy is a fair strategy.*

[10] Strictly speaking, this section implicitly assumes normalisation on LTSs. It improves presentation but is not technically needed.

Applying a round robin strategy *stg* on *LTS* gives a reduced LTS. Similar to [2,3], it can be shown that there exists a "representation mapping", which, for our case, maps an *entry point* to its *exit point*.

Let $\mathcal{MCC} = \{..., S_i, ...\}$ be the equivalence induced by the reflexive, symmetric, and transitive closure of A_i^τ transitions in *LTS*. Each member S_i is an equivalence class. Define the set of *stg* entry points on S_i as $ENT(S_i) \mathrel{\widehat{=}} \{s : S_i \mid s = s^0 \vee (stg(s_0 a_1 ... s_n, \tilde{k}) = (e, s) \wedge e \in A_v)\}$, and the set of *stg* exit points on S_i as $EXT(S_i) \mathrel{\widehat{=}} \{s_n : S_i \mid ((s_0 a_1 ... a_n s_n, e\tilde{k}) \in \mathrm{dom}\, stg \wedge fair\_loop(s_0 a_1 ... a_n s_n)) \vee stable(s_n, A_i^\tau)\}$. The set of *stg* exit points are exactly those states at which **rule 2b′** is activated or A_i^τ-stability is reached.

Proposition 6.2 *Given detachable A_i and a round robin strategy stg on LTS, if any execution of stg at any time enters $S_i \in \mathcal{MCC}$ with the intention to leave (i.e. having a pending e), then the exit point ($\in EXT(S_i)$) is uniquely determined by its entry point ($\in ENT(S_i)$).*

(**Representative function**) Let *ENT* and *EXT* be the union sets of entry and exit points for all $S_i \in \mathcal{MCC}$. Therefore there exists a representative function, $exit : ENT \to EXT$, mapping each entry point to its exit point. An exit point can fully represent all its entry points. If the exit point is A_i^τ-stable, then the set of outgoing transitions on the representative is exactly the set of outgoing transitions on the point. Otherwise, the set of outgoing transitions on the representative is exactly the set of A_v outgoing transitions combined with a τ self-loop.

Definition 6.3 [Reduction function] Given detachable A_i from *LTS*, function $chase^+(LTS, A_i)$ outputs another LTS, $(A_v, EXT, T', exit(s^0))$, where $T' = \{(s, e, s') : EXT \times A_v \times EXT \mid \exists\, s_i : S \bullet s \xrightarrow{e} s_i \wedge s' = exit(s_i)\} \cup \{(s, \tau, s) \mid s \in EXT \wedge \neg\, stable(s, A_i^\tau)\}$.

Therefore, we can adopt a scheme similar to that in [2] to implement $chase^+$ as an on-the-fly procedure integrated in refinement or model checking. Note also that round robin strategies are *local* strategies. That is, the definition only depends on the pending action and the top elements of the history and the exit points can be calculated by simply following the strategy. Therefore, the *exit* function need not be explicitly constructed. It enables a simpler and more efficient implementation of the $chase^+$ reduction procedure.

Theorem 6.4 (Preservation) *Δ is detachable from finite state LTS implies $chase^+(LTS, \Delta)$ is normalised and $chase^+(LTS, \Delta) \overset{SBDF}{=} LTS \setminus \Delta$.*

7 Compositional reduction

For the reduction technique of this paper to work effectively, it is preferable to represent all processes (including U-nondeterministic ones) in the form of $LTS^N \setminus \Delta$ rather than directly as unnormalised LTSs. Our philosophy is that if one is inquisitive enough on details, all the unaccounted-for choices in LTS, i.e. those due to τ-transitions or ambiguous transitions, can be accounted for by introducing some extra lvgi actions. This will not result in any loss of expressiveness, e.g. w.r.t.

SBDF models. Moreover, these lvgi choices need to remain so during the verification process, unless they are detachable, in which case they can be hidden and removed after reduction.

Therefore, a network of processes can be represented as $SC[\overrightarrow{LTS^N}]$, where SC is a "process context", and the following theorem can be applied on it.

Theorem 7.1 (Compositionality) [11] *LTS_1^N and LTS_2^N have untangled action sets U_1 and U_2 (respectively) implies $U_{\|}$ is untangled in $LTS_1^N \parallel LTS_2^N$, where $U_{\|} = (A_1 \cup A_2) \setminus ((A_1 \setminus U_1) \cup (A_2 \setminus U_2))$.*

Thus, our reduction works as follows:

(i) $SC[\overrightarrow{LTS^N}]$ can be transformed to $(\|[\overrightarrow{LTS'^N}]) \setminus \Delta$.

(ii) On each LTS'^N, find the maximal untangled subset of Δ, say U.

(iii) Use the Theorem 7.1 to calculate the global untangled action set $U_{\|}$ from $\overrightarrow{U}$.

(iv) Apply $chase^+$ on the global system and we have the final reduced system: $chase^+(\|[\overrightarrow{LTS''^N}], U_{\|}) \setminus (\Delta \setminus U_{\|})$ [12].

Preliminary experiment. The CSP check in [23] was tested on the arbiter cell. It took a fraction of a second to correctly identify that the set of maximal untangled subset is $A \setminus \{a1+, a2+\}$. Theorem 7.1 then showed that all the actions in the tree arbiter, except those of $a1+$ and $a2+$, are untangled.

Since $chase^+$ is not available in FDR2 yet, *chase* is used instead to reduce the state space. Fortunately, this is correct due to the fact that a tree arbiter remains a divergence-free system after the untangled actions are hidden. Actually nested *chase* were applied along the tree of arbiter cells.

We checked the system using FDR2. The results are very encouraging compared to previous works [1,10]. The checking time is nearly linear in the size of the tree arbiter. More intriguingly the memory used is negligible (below 100MByte) and is sub-linear relative to the tree size. Thus, it is fair to say that the state explosion has been avoided.

8 Conclusion

Relative to previous works, the merits of the current work are summarised as follows:

- Our reduction technique is compositional and places minimal restrictions on the synchronisation potential of processes.
- It gives an accurate treatment on divergence despite the interference between divergence and compositionality. That is, a divergent process can delay other parallel processes indefinitely. Our solution is to keep lvgi actions visible and use fairness to guide state space traversal out of unprogressing loops.
- It uses a weakest possible failure equivalence and thus has advantages in reduction.

[11] Note that the maximality of untangled action sets is not necessarily preserved in this theorem.

[12] Another, potentially more efficient, approach is to push the hiding of $U_{\|}$ downwards along the parallel composition hierarchy as much as possible, and then apply nested $chase^+$ layer by layer.

- A hierarchy of partial determinacy properties are identified, e.g. untangledness, detachability and U-determinism. They can be of independent interests. For instance, it seems possible that any CSP process equals a (possibly infinite) non-deterministic choice on a set of U-deterministic processes [19,15].

Acknowledgements We are gratful to A.W. Roscoe for reading an earlier draft of this paper and giving valuable suggestions, and to Henri Hansen for explaining their work on divergence-preserving operational determinism.

References

[1] R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer and S. K. Rajamani. Partial-Order Reduction in Symbolic State Space Exploration. CAV 1997: 340-351.

[2] S. Blom. Partial τ-confluence for Efficient State Space Generation. Technical Report SEN-R0123, CWI, Amsterdam, 2001.

[3] S. Blom and J. van de Pol. State Space Reduction by Proving Confluence. CAV 2002: 596-609.

[4] A. Davis and S. M. Norwick. *An Introduction to Asynchronous Circuit Design*. The Encyclopedia of Computer Science and Technology (vol 38), Marcel Dekker, New York, 1998.

[5] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1993.

[6] Formal Systems (Europe) Ltd. *Failures-Divergence Refinement: FDR2 User Manual*, 1999.

[7] J.F. Groote and J.C. van de Pol. State space reduction using partial tau-confluence. MFCS 2000, LNCS 1893.

[8] J. F. Groote and M. P. Sellink. Confluence for Process Verification. CONCUR 1995, LNCS 962.

[9] H. Hansen and A. Valmari. Operational Determinism and Fast Algorithms. CONCUR 2006.

[10] K. L. McMillan. Trace Theoretic Verification of Asynchronous Circuits Using Unfoldings. CAV 1995: 180-195.

[11] G. J. Pace, F. Lang and R. Mateescu. Calculating-Confluence Compositionally. CAV 2003: 446-459.

[12] D. Peled. Partial Order Reduction: Linear and Branching Temporal Logics and Process Algebras. Proceedings of POMIV'96, DIMACS Series Vol. 29, AMS, 1997.

[13] A. Puhakka and A. Valmari. Weakest-Congruence Results for Livelock-Preserving Equivalences. Proceedings of CONCUR '99, LNCS 1664.

[14] Y. S. Ramakrishna and S. A. Smolka. Partial-Order Reduction in the Weak Modal Mu-Calculus. CONCUR 1997, LNCS 1243.

[15] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 1998.

[16] A. W. Roscoe. Seeing beyond divergence. Proceedings of "Symposium on the Occasion of 25 years of CSP", London, July 2004, LNCS 3525.

[17] A. W. Roscoe. The pursuit of buffer tolerance. Unpublished manuscript, 2005.

[18] A. W. Roscoe. Confluence thanks to extensional determinism. Bertinoro meeting on Concurrency, BRICS 2005.

[19] A. W. Roscoe. Personal communication, March 2006.

[20] A. Valmari. The Weakest Deadlock-Preserving Congruence. Information Processing Letters 53 (1995) 341-346.

[21] A. Valmari. Stubborn Set Methods for Process Algebras. POMIV'96, DIMACS Series Vol. 29, AMS, 1997.

[22] X. Wang, M. Kwiatkowska. On process-algebraic verification of asynchronous circuits. ACSD 2006.

[23] X. Wang, M. Kwiatkowska. Compositional state space reduction using untangled actions. Tech. Rep. CSR-06-7, School of computer science, University of Birmingham.

[24] G. Winskel. Event structures. In Advances in Petri Nets 1986, Part II; 1987.

[25] A. Yakovlev, A. Petrov and L. Lavagno. A Low Latency Asynchronous Arbitration Circuit. IEEE Trans. on VLSI Systems, vol 2, no 3, 1994.

On Symbolic Verification of Weakly Extended PAD

Ahmed Bouajjani[a,1], Jan Strejček[b,2], and Tayssir Touili[a,3]

[a] *LIAFA, University of Paris 7, France*

[b] *LaBRI, Univeristy of Bordeaux 1, France*

Abstract

We consider the verification problem of a class of infinite-state systems called wPAD. These systems can be used to model programs with (possibly recursive) procedure calls and dynamic creation of parallel processes. They correspond to PAD models extended with an acyclic finite-state control unit, where PAD models can be seen as combinations of prefix rewrite systems (pushdown systems) with context-free multiset rewrite systems (synchronization-free Petri nets). Recently, we have presented symbolic reachability techniques for the class of PAD based on the use of a class of unranked tree automata. In this paper, we generalize our previous work to the class wPAD which is strictly larger than PAD. This generalization brings a positive answer to an open question on decidability of the model checking problem for wPAD against EF logic. Moreover, we show how symbolic reachability analysis of wPAD can be used in (under) approximate analysis of Synchronized PAD, a (Turing) powerful model for multithreaded programs (with unrestricted synchronization between parallel processes). This leads to a pragmatic approach for detecting the presence of erroneous behaviors in these models based on the bounded reachability paradigm where the notion of bound considered here is the number of synchronization actions.

Keywords: rewrite systems, infinite-state systems, symbolic reachability analysis, model checking

1 Introduction

Reasoning about software systems requires the consideration of powerful models which are in general infinite-state, i.e., they may have an infinite number of reachable configurations. Sources of complexity, and of infinity of the state space, may be related to either *data manipulation* such as the use of variables over infinite data domains, dynamic and unbounded-size data structures, etc, or to *complex control primitives* such as procedures calls, (unbounded) dynamic creation of concurrent processes, etc. One popular approach to handle this complexity is to combine abstraction methods with model-checking. Techniques such as predicate abstraction allows to deal with aspects such as data manipulation and to generate abstract

[1] Email: abou@liafa.jussieu.fr

[2] Partly supported by the research centre Institute for Theoretical Computer Science (ITI), project No. 1M0545. Email: strejcek@labri.fr

[3] Email: touili@liafa.jussieu.fr

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

models over finite data domains. Then, the so obtained abstract models can be analyzed automatically using model checking algorithms, provided that such algorithms exist for the considered class of abstract models. This is the case obviously when abstract models are finite-state. However, as said above, in order to take into account complex control primitives such as procedure calls and process creation, finite state models are not expressive enough. For instance, in the case of sequential programs with recursive procedure calls, the needed abstract models are (unbounded-stack) pushdown systems, and for programs with dynamic creation of communicating finite-state processes, natural models are (unbounded) Petri nets. Fortunately, there exist several algorithmic techniques (e.g., reachability analysis, model-checking) which have been developed for the analysis and the verification of these infinite-state models.

In this paper, we consider the case of programs which may contain both (recursive) procedure calls and dynamic creation of processes (threads). One possible approach to model such systems is to combine pushdown systems with Petri nets. This corresponds to the use of *Process Rewrite Systems* (PRS) introduced in [18]. These models can be seen indeed as combinations of prefix rewrite systems and multiset rewrite systems. The relevance of PRS in program modeling have been discussed for instance in [9,10,8,2,3]. Subclasses of PRS which are of particular interest for program modeling are for instance the class of PA processes, and the larger class of PAD processes generalizing both PA and pushdown processes and corresponding to synchronization-free PRS (i.e., models where parallel composition is not allowed in the left-hand-side of the rewrite rules). Processes in these classes allow indeed to model systems with procedure calls and parbegin-parend blocks (i.e., launching a number of parallel threads, and wait for their termination before proceeding). PAD allow in addition return values from sequential procedure calls.

Mayr has shown that the *reachability problem* (whether a given state is reachable from another given state) for PRS is decidable using a reduction to the reachability problem of Petri nets [18]. To get practical verification algorithms, symbolic reachability algorithms have been investigated for significant subclasses of PRS such as PA [17,10] and PAD [2,3]. These algorithms use (various kinds of) tree automata to represent (regular) infinite sets of configurations (i.e., process terms). In particular, we have provided in [3] a generic construction allowing to compute the set of (forward or backward) reachable configurations of any subclass of PRS built from the combination of prefix rewrite systems with an effectively semilinear class of multiset rewrite systems (i.e., a class of systems for which reachability sets are always semilinear and effectively computable). We have shown that this leads to a symbolic reachability analysis algorithm for PAD processes in a certain normal form.

The PRS formalism is not Turing powerful due to a subtle restriction on the way synchronization is done between parallel processes. Roughly speaking, the semantics of PRS implies that synchronization can only be allowed between parallel processes with empty stacks.

In order to extend the modeling power of PRS, one approach is to add synchronization by rendez-vous (à la CCS), which leads to a Turing powerful model called *synchronized PRS* [22]. Similarly, PAD can be extended to *synchronized PAD* (which is also a Turing powerful model). Approximate analysis algorithms for these

models using abstraction techniques have been proposed in [22].

Another approach for enhancing the modeling power of PRS (and PAD) consists in adding global control states. The new models, called sePRS [12], can be seen as parallel product of a PRS with a finite-state automaton representing a global control. Obviously, sePRS are Turing powerful since they allow communication between recursive parallel processes through the global control state. However, if the structure of the control automaton is *weak*, which means that all its loops are self-loops, then it can be proved that the obtained models, called wPRS, have a decidable reachability problem [13] (the proof employs decidability of the reachability problem for Petri nets). Similarly, if we add control states to PAD processes, we obtain Turing powerful models, but the extension of PAD with weak control automata leads to models, called wPAD, having a decidable reachability problem, and interestingly, which can be proven to be strictly more powerful (w.r.t. strong bisimulation) than PAD [14].

In this paper we extend the results on symbolic reachability analysis presented in [3]. While [3] deals only with PAD processes in a certain normal form (now called *canonic PAD*), here we show that the set of reachability states are computable and effectively representable even for (general) wPAD systems. To do this, we employ symbolic representations based on so-called *commutative-hedge automata* (CH-automata), allowing to define sets of process terms modulo the associativity of sequential composition, and the associativity-commutativity of the parallel composition. We show that these representations are effectively closed under the computation of the $post^*$ and pre^* images (i.e., computation of all successors and all predecessors) for wPAD, as well as under the $post$ and pre images (i.e., computation of immediate successors and predecessors) for the *whole class of wPRS*.

Further, we solve the *global model-checking* problem of wPAD against the EF logic. We consider a variant of EF logic which generalizes the standard action-based EF logic by the use of atomic propositions corresponding to (potentially infinite) sets of configurations which are definable using CH-automata. We prove that for every formula in this logic, it is possible to construct a (CH-automata based) representation of the set of all configurations (in a given wPAD) satisfying this formula. This result closes an open problem formulated in [15] concerning the model-checking problem of wPAD. Notice that global model-checking is a more general problem than deciding whether a given configuration satisfies a given formula.

Our results concerning symbolic reachability analysis of wPAD can be used in the analysis of *synchronized PAD* (SPAD) with a bounded number of synchronizations. This leads to an approximate analysis procedure for SPAD based on computing *under* approximations of their reachability sets by considering only reachable configurations up to some fixed number of synchronizations. Such approximate analysis method for SPAD can be used in practice to establish the *existence* of erroneous behaviors, following the approach advocated in [19]. It constitutes a complementary approach to the abstract analysis (provided for the same models in [22]), which is based on considering *upper* approximations of the set of possible behaviors and which is useful for establishing the *absence* of erroneous behaviors.

The under approximation mentod for SPAD as well as proofs of Theorems 2.1 and 4.1 can be found in the full version of this paper [1].

2 Preliminaries

2.1 Process terms

Let $Const = \{X, \ldots\}$ be a set of *process constants*. For every $C \subseteq Const$, the set T_C of *process terms* over C is defined by the abstract syntax $t ::= 0 \mid X \mid t \odot t \mid t \| t$, where 0 is the *idle term*, $X \in C$ is a process constant; and $\odot$ and $\|$ mean *sequential* and *parallel compositions* respectively.

We use ω to denote in a generic way $\odot$ or $\|$. We denote by $\overline{\omega}$ the operator $\odot$ (resp. $\|$) if $\omega = \|$ (resp. $\omega = \odot$). Process terms are considered modulo the following algebraic properties: associativity of $\odot$, associativity and commutativity of $\|$, and neutrality of 0 w.r.t. both $\odot$ and $\|$, i.e. $0 \odot t = t \odot 0 = t \| 0 = t$. Let $\simeq$ be the equivalence relation on T induced by these properties.

We distinguish four *classes of process terms* as:

1 – terms consisting of a single process constant only, in particular $0 \notin 1$,

S – *sequential* terms - terms without parallel composition, e.g. $X \odot Y \odot Z$,

P – *parallel* terms - terms without sequential composition, e.g. $X \| Y \| Z$,

G – *general* terms - terms without any restrictions, e.g. $(X \odot (Y \| Z)) \| W$.

Process terms in *canonical form* are terms t defined by:

$$t ::= 0 \mid s \mid p$$
$$s ::= X \mid p_1 \odot p_2 \odot \ldots \odot p_n,\ n \geq 2$$
$$p ::= X \mid s_1 \| s_2 \| \ldots \| s_n,\ n \geq 2$$

It can easily be seen that every term has an $\simeq$-equivalent term in canonical form. In the following we work with terms in canonical form.

Term t is called *seq-term* if $t = 0$, or $t = X$ for a constant X, or $t = p_1 \odot p_2 \odot \ldots \odot p_n$ where $n \geq 2$. In the last case, the term is also called $\odot$*-rooted term*. Further, t is called *flat seq-term* if $t = X_1 \odot X_2 \odot \ldots \odot X_n$ for $n \geq 0$ (the case $n = 0$ corresponds to the term 0, and the case $n = 1$ corresponds to a process constant X). By analogy we define *par-terms*, $\|$*-rooted terms*, and *flat par-terms*.

2.2 Process Rewrite Systems and weak extension

Let $M = \{o, p, q, \ldots\}$ be an ordered set of *control states* and $Act = \{a, b, c, \ldots\}$ be a set of *actions*. Let $\alpha, \beta \in \{1, S, P, G\}$ be classes of process terms such that $\alpha \subseteq \beta$. An *(α, β)-wPRS (weakly extended process rewrite system)* R is a finite set of *rewrite rules* of the form $(p, t_1) \overset{a}{\hookrightarrow} (q, t_2)$, where $t_1 \in \alpha$, $t_1 \neq 0$, $t_2 \in \beta$, $p, q \in M$, $p \leq q$, and $a \in Act$. By $M(R)$, $Const(R)$, and $Act(R)$ we denote sets of control states, process constants, and actions occurring in rewrite rules of R.

An (α, β)-wPRS R induces a labelled transition system the states of which are pairs (p, t) such that $p \in M(R)$ is a control state and $t \in \beta$ is a process term over $Const(R)$. The transition relation $\rightarrow_R$ is the least relation satisfying the following inference rules:

$$\frac{((p, t_1) \overset{a}{\hookrightarrow} (q, t_2)) \in R}{(p, t_1) \overset{a}{\rightarrow}_R (q, t_2)} \qquad \frac{(p, t_1) \overset{a}{\rightarrow}_R (q, t_2)}{(p, t_1 \| t) \overset{a}{\rightarrow}_R (q, t_2 \| t)} \qquad \frac{(p, t_1) \overset{a}{\rightarrow}_R (q, t_2)}{(p, t_1 \odot t) \overset{a}{\rightarrow}_R (q, t_2 \odot t)}$$

We extend the transition relation to finite words over Act in a standard way. The reflexive and transitive closure of $\rightarrow_R$ is denoted by $\stackrel{*}{\rightarrow}_R$. To shorten our notation we write pt in lieu of (p,t).

An (α,β)-wPRS where $M(R)$ is a singleton is called *(α,β)-PRS (process rewrite system)*. In such systems we omit the single control state from rules and states.

Instead of (S,G)-PRS, (S,G)-wPRS, (G,G)-PRS, and (G,G)-wPRS we use more readable names PAD, wPAD, PRS, and wPRS respectively. Let us note that the classes PAD and wPAD subsume widely known models of infinite-state systems as *pushdown processes* (PDA), *basic parallel processes* (BPP), and *process algebras* (PA). The classes PRS and wPRS subsume also *Petri nets* (PN). More information about expressiveness of (α,β)-wPRS and (α,β)-wPRS can be found in [14,13].

Given a state pt of a wPRS R, we define

$$Post_R(pt) = \{p't' \mid pt \stackrel{a}{\rightarrow}_R p't' \text{ for some } a\} \qquad Post^*_R(pt) = \{p't' \mid pt \stackrel{*}{\rightarrow}_R p't'\}$$
$$Pre_R(pt) = \{p't' \mid p't' \stackrel{a}{\rightarrow}_R pt \text{ for some } a\} \qquad Pre^*_R(pt) = \{p't' \mid p't' \stackrel{*}{\rightarrow}_R pt\}$$

The sets $Post^*_R(pt)$ and $Pre^*_R(pt)$ are called *(forward and backward) reachability sets*. The sets $Post_R(pt)$ and $Pre_R(pt)$ are called *1-step (forward and backward) reachability sets*. These definitions and notations can be extended to sets of states in the obvious manner.

2.3 Canonic PRS

A *canonic PRS* R is a set of rewrite rules of the forms:

$$X_1 \odot X_2 \odot \ldots \odot X_n \stackrel{a}{\hookrightarrow} Y_1 \odot Y_2 \odot \ldots \odot Y_m \tag{1}$$
$$X_1 \| X_2 \| \ldots \| X_n \stackrel{a}{\hookrightarrow} Y_1 \| Y_2 \| \ldots \| Y_m \tag{2}$$

where $n,m \geq 0$. Rules of the form (1) and (2) are called *$\odot$-rules* and *$\|$-rules* respectively. By R_ω we denote the set of all ω-rules of R. Note that the sets $R_\|$ and $R_\odot$ do not have to be disjoint as some rules (e.g. $X \stackrel{a}{\hookrightarrow} Y$) are of both types. Let $\alpha,\beta \in \{1,S,P,G\}$ be classes of process terms. A canonic PRS is called *canonic (α,β)-PRS* if every rule $t_1 \stackrel{a}{\hookrightarrow} t_2$ of R satisfies $t_1 \in \alpha$ and $t_2 \in \beta$. Finally, *canonic PAD* stands for canonic (S,G)-PRS.

Note that a canonic PRS does not have to be a PRS as we allow rules with 0 on the left-hand side. Further, the definition of canonic (α,β)-PRS does not require that $\alpha \subseteq \beta$. The meaning of $Const(R), \rightarrow_R, Post_R, Pre_R, \ldots$ remains the same.

Given a canonic (α,β)-PRS R, by R^{-1} we denote the canonic (β,α)-PRS with rules obtained by swapping the left-hand and right-hand sides of the rules of R. Notice that for every set of process terms L, $Pre_R(L) = Post_{R^{-1}}(L)$ and $Pre^*_R(L) = Post^*_{R^{-1}}(L)$.

The problem of computing reachability sets of PRS systems can be transformed into the same problem for canonic PRS using the following theorem. The proof of this theorem (available in [1]) employs a variant of the standard construction given in [18]. However, our theorem differs from the one of [18] in several aspects. In particular, (1) we transform an (α,β)-PRS into a canonic (α,β)-PRS, which is not the case of Mayr's transformation, and (2) in contrast to the original theorem

in [18], our theorem states that the same transformation of R works for *all* terms over a given set of process constants.

A *term substitution* h is a function on process terms satisfying $h(0) = 0$ and $h(t_1 \,\omega\, \ldots \,\omega\, t_n) = h(t_1)\,\omega\, \ldots\, \omega\, h(t_n)$ for all finite sequences $t_1, \ldots, t_n$ of terms and for both $\omega = \odot, \|$. In other words, a term substitution is fully specified by its values on process constants. We say that a term subsitution h is *finite* if the set $\{X \mid h(X) \neq X\}$ of process constants is finite.

Theorem 2.1 *For every (α, β)-PRS system R and every set of process constants C we can construct a canonic (α, β)-PRS system R' and a finite term substitution h, such that for every t_1, t_2 over $C \cup Const(R)$ and every $a \in Act(R)$ we have:*

(i) $t_1 \xrightarrow{a}_R t_2$ *iff there exists* t'_1, t'_2 *satisfying* $h(t'_1) = t_1$, $h(t'_2) = t_2$, *and* $t'_1 \xrightarrow{a}_{R'} t'_2$,

(ii) $t_1 \xrightarrow{*}_R t_2$ *iff there exists* t'_1, t'_2 *satisfying* $h(t'_1) = t_1$, $h(t'_2) = t_2$, *and* $t'_1 \xrightarrow{*}_{R'} t'_2$.

3 Automata-based symbolic representations

In order to perform reachability analysis of PRS, we need representation structures for (infinite) sets of process terms. For this purpose, we use a class of tree-automata, called *commutative hedge automata* [3], which recognize sets of trees modulo associativity / associativity-commutativity. These automata extend both (1) bottom-up tree automata over ranked alphabets [6], and (2) hedge automata recognizing sets of undounded width trees [4].

3.1 Preliminaries

Presburger arithmetic is the first order logic of integers with addition and linear ordering. Given a formula φ, we denote by $FV(\varphi)$ the set of its free variables. Let $FV(\varphi) = \{x_1, \ldots, x_n\}$. Then, a vector $\boldsymbol{u} = (u_1, \ldots, u_n) \in \mathbb{Z}^n$ satisfies φ, written $\boldsymbol{u} \models \varphi$, if $\varphi(\boldsymbol{u}) = \varphi[x_i \leftarrow u_i]$ is true. Each formula φ defines a set of integer vectors $[\![\varphi]\!] = \{\boldsymbol{u} \in \mathbb{Z}^n \mid \boldsymbol{u} \models \varphi\}$. Presburger formulas define *semilinear sets* of integer vectors, i.e., finite union of sets of the form $\{\boldsymbol{x} \in \mathbb{Z}^n \mid \exists k_1, \ldots, k_n \in \mathbb{Z}, \boldsymbol{x} = \boldsymbol{v}_0 + k_1\boldsymbol{v}_1 \cdots + k_n\boldsymbol{v}_m\}$, where $\boldsymbol{v}_i \in \mathbb{Z}^n$, for $1 \leq i \leq m$ (see [11]).

Given a word w over an alphabet $\Sigma = \{a_1, \ldots, a_n\}$, the *Parikh image* of w, denoted $Parikh(w)$, is the vector $(|w|_{a_1}, \ldots, |w|_{a_n})$. This definition can be generalized to sets of words (languages) over Σ in the obvious manner.

As usual, a set of words is *regular* if it is definable by a finite-state automaton. The notion of regularity can be transfered straightforwardly to sets of flat seq-terms. Similarly, the notion of semilinearity can be transfered to sets of flat par-term by associating with a term $X_1 \| \cdots \| X_n$ the vector $Parikh(X_1 \cdots X_n)$.

In the sequel, we will represent by γ a *constraint* which is either a regular language or a Presburger formula. We say that a word $w = a_1 a_2 \ldots a_n$ *satisfies* the constraint γ if $w \in \gamma$ (resp. $Parikh(w) \models \gamma$) when γ is a language (resp. a formula).

3.2 Commutative Hedge Automata

Let $\Sigma = \Sigma' \cup \Sigma_A$ be a finite alphabet, where Σ' is a ranked alphabet, and Σ_A is a finite set of associative operators. We assume that Σ' and Σ_A are disjoint. For

$k \geq 0$, let Σ_k denote the set of elements of Σ' of rank k.

3.2.1 Σ-Terms:
Let $\mathcal{X}$ be a fixed countable set of variables $\{x_1, x_2, \ldots\}$. The set $T_\Sigma[\mathcal{X}]$ of Σ-terms over $\mathcal{X}$ is the smallest set such that:

- $\Sigma_0 \cup \mathcal{X} \subseteq T_\Sigma[\mathcal{X}]$,
- for $k \geq 1$, if $f \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma[\mathcal{X}]$, then $f(t_1, \ldots, t_k) \in T_\Sigma[\mathcal{X}]$,
- if $f \in \Sigma_A$, $t_1, \ldots, t_n \in T_\Sigma[\mathcal{X}]$ for some $n \geq 1$, and $root(t_i) \neq f$ for every $1 \leq i \leq n$, then $f(t_1, \ldots, t_n) \in T_\Sigma[\mathcal{X}]$, where $root(\sigma) = \sigma$ if $\sigma \in \Sigma_0 \cup \mathcal{X}$, and $root\big(g(u_1, \ldots, u_m)\big) = g$.

Note that if $f \in \Sigma_A$, we only consider terms of the form $f(t_1, \ldots, t_n)$ such that for every i, the root of t_i is different from f. Indeed, since f is associative, $f\big(t_1, \ldots, t_{i-1}, f(u_1, \ldots, u_m), t_{i+1}, \ldots, t_n\big)$ is equivalent to the term $f(t_1, \ldots, t_{i-1}, u_1, \ldots, u_m, t_{i+1}, \ldots, t_n)$.

Terms without variables are called *ground terms*. Let T_Σ be the set of ground terms of $T_\Sigma[\mathcal{X}]$. A term t in $T_\Sigma[\mathcal{X}]$ is *linear* if each variable occurs at most once in t. A *context* C is a linear term of $T_\Sigma[\mathcal{X}]$. Let $t_1, \ldots, t_n$ be terms of T_Σ, then $C[t_1, \ldots, t_n]$ denotes the term obtained by replacing in the context C the occurrence of the variable x_i by the term t_i, for each $1 \leq i \leq n$.

3.2.2 Definition of CH-automata:
Let us consider that $\Sigma_A = \Sigma'_A \cup \Sigma'_{AC}$ where Σ'_{AC} is a set of associative and commutative operators. We assume that Σ'_A and Σ'_{AC} are disjoint. Then, a CH-automaton is a tuple $\mathcal{A} = (Q, \Sigma, F, \Delta)$ where:

- Q is a union of disjoint finite sets of states $Q' \cup \bigcup_{f \in \Sigma_A} Q_f$,
- $F \subseteq Q$ is a set of final states,
- Δ is a set of rules of the form:

(i) $a \to q$, where $q \in Q', a \in \Sigma_0$,
(ii) $f(q_1, \ldots, q_k) \to q$, where $f \in \Sigma_k, q \in Q'$, and $q_i \in Q$,
(iii) $q \to q'$, where $(q, q') \in Q' \times Q' \cup \bigcup_{f \in \Sigma_A} Q_f \times Q_f$,
(iv) $f(Reg) \to q$, where $f \in \Sigma'_A$, $Reg \subseteq (Q \setminus Q_f)^*$ is a regular language given by a finite-state automaton, and $q \in Q_f$,
(v) $f(\varphi) \to q$, where $f \in \Sigma'_{AC}$, $q \in Q_f$, and φ is a Presburger formula such that $FV(\varphi) = \{x_q \mid q \in Q \setminus Q_f\}$.

We define a *move relation* $\to_\Delta$ between ground terms in $T_{\Sigma \cup Q}$ as follows: for every two terms t and t', we have $t \to_\Delta t'$ iff there exist a context C and a rule $r \in \Delta$ such that $t = C[s]$, $t' = C[s']$, and:

- $r = a \to q$, with $s = a$ and $s' = q$, or
- $r = q \to q'$, with $s = q$ and $s' = q'$, or
- $r = f(q_1, \ldots, q_k) \to q$, with $s = f(q_1, \ldots, q_k)$ and $s' = q$, or
- $r = f(Reg) \to q$, with $f \in \Sigma'_A$, $s = f(q_1, \ldots, q_n)$, $q_1 \cdots q_n \in Reg$, and $s' = q$, or
- $r = f(\varphi) \to q$, with $f \in \Sigma'_{AC}$, $s = f(q_1, \ldots, q_n)$, $Parikh(q_1 \cdots q_n) \models \varphi$, and

$s' = q$.

Let $\xrightarrow{*}_\Delta$ denote the reflexive-transitive closure of $\rightarrow_\Delta$. A ground term $t \in T_\Sigma$ is *accepted by a state* q if $t \xrightarrow{*}_\Delta q$. Let $L_q = \{t \in T_\Sigma \mid t \xrightarrow{*}_\Delta q\}$. A ground term $t \in T_\Sigma$ is *accepted by the automaton* $\mathcal{A}$ if it is accepted by some final state $q \in F$. The CH-language of $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of all ground terms accepted by $\mathcal{A}$.

We have the following fact [5,16,20,21,3]:

Theorem 3.1 *The class of languages recognized by CH-automata is effectively closed under boolean operations, term substitutions and inverse of finite term substitutions. Moreover, the emptiness problem of CH-automata is decidable.*

3.3 CH-automata for PRS process terms

We consider process terms as trees and use CH-automata to represent sets of such trees. Indeed, for any finite set $C \subseteq Const$, the set T_C of process terms can be seen as the set of Σ-terms T_Σ where $\Sigma_0 = \{0\} \cup C$, $\Sigma'_A = \{\odot\}$, and $\Sigma'_{AC} = \{\|\}$.

Sets of process terms are recognized by CH-automata $\mathcal{A} = (Q, \Sigma, F, \Delta)$ such that (1) Q is the disjoint union $Q = Q' \cup Q_\odot \cup Q_\|$ where Q' is itself the disjoint union $Q' = Q_0 \cup Q_-$, and (2) the rules in Δ are of the form: (a) $X \rightarrow q$, where $q \in Q_-$, $X \in Const$, (b) $0 \rightarrow q$, where $q \in Q_0$, (c) $q \rightarrow q'$, where $(q, q') \in (Q_0)^2 \cup (Q_-)^2 \cup (Q_\odot)^2 \cup (Q_\|)^2$, (d) $\odot(Reg) \rightarrow q$, where $Reg \subseteq \big(Q \setminus (Q_\odot \cup Q_0)\big)^*$ is a regular language and $q \in Q_\odot$, and (e) $\|(\varphi) \rightarrow q$, where $q \in Q_\|$ and φ is a Presburger formula such that $FV(\varphi) = \{x_q \mid q \in Q \setminus (Q_\| \cup Q_0)\}$.

In other words, the states in $Q_\odot$ (resp. $Q_\|$) recognize trees whose root is $\odot$ (resp. $\|$). The states in Q_- recognize constants in C, and the states in Q_0 recognize 0.

4 Computing 1-step reachability sets for canonic PRS

Let us consider a canonic PRS $R = R_\odot \cup R_\|$ and let $\mathcal{A} = (Q, \Sigma, F, \Delta)$ be a CH-automaton recognizing a set L of process terms. We show that the sets $Post_R(L)$ and $Pre_R(L)$ are effectively representable and computable by CH-automata.

For a given canonic PRS R' and a given set of terms L_1, we write $R'(L_1)$ as an abbreviation for $Post_{R'}(L_1)$. In the following we use the fact that given a regular set L_2 of flat seq-terms, the set $R'_\odot(L_2)$ is again regular and easily constructible. The same holds for any semilinear sets L_3 of flat par-terms and $R'_\|(L_3)$.

We construct a CH-automaton $\mathcal{A}' = (\widetilde{Q}, \Sigma, \widetilde{F}, \widetilde{\Delta})$ which recognizes $R(L)$, where $\widetilde{Q}$ is the set of states, $\widetilde{F}$ is the set of final states, and $\widetilde{\Delta}$ is the set of rules. Let C be a finite set of process constants such that $C \supseteq Const(R)$ and $L \subseteq T_C$.

4.1 The set of states

The set of states $\widetilde{Q}$ includes the set of states Q of $\mathcal{A}$ and contains new states q_X, which are assumed to accept precisely the singletons $\{X\}$ (i.e., $L_{q_X} = \{X\}$), for each $X \in C$. Let Q_R be the set of states $Q_R = \{q_X \mid X \in C\}$. In addition, the set $\widetilde{Q}$ contains states which recognize the set $R(L_q)$ of *immediate successors* of terms in L_q for each $q \in Q \cup Q_R$. In order to ensure (during the construction) that

the recognized trees are always in canonical form, we need to partition the sets of recognized trees according to their types (given by their root).

We associate with each $q \in Q \cup Q_R$ different states $(q,-), (q,0), (q,\odot)$, and $(q,\|)$ recognizing immediate successors of terms in L_q which are respectively constants in C, null (equal to 0), $\odot$-rooted terms, and $\|$-rooted terms.

Let $Q = Q_0 \cup Q_- \cup Q_\odot \cup Q_\|$. We consider that the set $\widetilde{Q}$ is equal to the union of the following sets: (1) $\widetilde{Q}_0 = Q_0 \cup \{(q,0) \mid q \in Q \cup Q_R\}$, (2) $\widetilde{Q}_- = Q_- \cup Q_R \cup \{(q,-) \mid q \in Q \cup Q_R\}$, and (3) $\widetilde{Q}_\omega = Q_\omega \cup \{(q,\omega) \mid q \in Q \cup Q_R\}$, for $\omega \in \{\odot, \|\}$. Moreover, we consider that $\widetilde{F} = \{(q,-), (q,0), (q,\odot), (q,\|) \mid q \in F\}$.

4.2 Rewrite system over the alphabet of states

Rules in CH-automata (of the forms $\omega(\gamma) \to q$) involve constraints on sequences of *states*, whereas the systems $R_\odot$ and $R_\|$ are defined over the alphabet of process constants. Therefore, we define the systems $S_\odot = \alpha(R_\odot)$ and $S_\| = \alpha(R_\|)$ where α is the substitution such that $\alpha(X) = q_X$, for every $X \in C$ (extended in the standard way to terms, rules, and sets of rules).

4.3 The set of transition rules

The set $\widetilde{\Delta}$ is defined as the smallest set of transition rules which (1) contains Δ, (2) contains the set of rules $X \to q_X$ for every $X \in \mathit{Const}$, and (3) is such that:

(β_1) **Closure rules: successors of process constants and 0:**
(a) If $X \xrightarrow{*}_\Delta q$, then $\omega\big(S_\omega(q_X)\big) \to (q,\omega) \in \widetilde{\Delta}$,
(b) If $0 \xrightarrow{*}_\Delta q$, then $\omega\big(S_\omega(0)\big) \to (q,\omega) \in \widetilde{\Delta}$.
The rule (a) says that if X is in L_q, then all its immediate ω-successors obtained by applying once the system R_ω are also immediate successors of L_q. The rule (b) says the same thing for successors of 0.

(β_2) **Closure rule: successors of ω-rooted terms:** If $\omega(\gamma) \to p \in \Delta$, then $\omega\big(S_\omega(\sigma(\gamma))\big) \to (p,\omega) \in \widetilde{\Delta}$, where σ is the substitution such that $\forall q \in Q \cup Q_R$, $\sigma(q) = \{q\} \cup \{q_X \mid X \xrightarrow{*}_\Delta q\} \cup \{0 \mid 0 \xrightarrow{*}_\Delta q\}$.
This rule says that if $\omega(X_1,\ldots,X_n) \in L_p$ and $\omega(X'_1,\ldots,X'_m) \in R_\omega\big(\omega(X_1,\ldots,X_n)\big)$, then $\omega(X'_1,\ldots,X'_m)$ is a ω-successor of L_p.

(β_3) **Propagation rule:** If $\omega(\gamma) \to p \in \Delta$, then $\omega\big(E_\omega(\gamma)\big) \to (p,\omega) \in \widetilde{\Delta}$, where E is a canonic PRS defined as $E = \{q \hookrightarrow (q,-), q \hookrightarrow (q,\odot), q \hookrightarrow (q,\|)\}$.
The rule says that if $\odot(t_1,\ldots,t_n) \in L_p$ and t'_1 is a successor of t_1, then $\odot(t'_1,\ldots,t_n)$ is a successor of L_p. Moreover, if $\|(t_1,\ldots,t_n) \in L_p$ and t'_i is a successor of t_i, then $\|(t_1,\ldots,t'_i,\ldots,t_n)$ is a successor of L_p.
Note that we need to distinguish between $E_\|(\gamma)$ and $E_\odot(\gamma)$ to ensure that the prefix-rewrite strategy of the $\odot$ is correctly taken into account.

(β_4) **Term flattening rules:**
(a) If $\omega(\gamma) \to (q,\omega) \in \widetilde{\Delta}$ and $q' \in \gamma$, then $q' \to (q,-) \in \widetilde{\Delta}$ if $q' \in \widetilde{Q}_-$, and $q' \to (q,\overline{\omega}) \in \widetilde{\Delta}$ if $q' \in \widetilde{Q}_{\overline{\omega}}$.
(b) If $\omega(\gamma) \to (q,\omega) \in \widetilde{\Delta}$ and $0 \in \gamma$, then $0 \to (q,0) \in \widetilde{\Delta}$.
The rules say that if $\omega(t)$ is a successor of L_q, then t is also a successor of L_q.

Theorem 4.1 *For every canonic PRS R and every CH-automaton $\mathcal{A}$, we have $Post_R(L(\mathcal{A})) = L(\mathcal{A}')$.*

The proof can be found in [1]. As $Pre_R(L) = Post_{R^{-1}}(L)$, the previous construction can also be used to compute 1-step *backward* reachability sets.

5 Computing reachability sets for PAD and wPAD

In this section, we solve the problem of computing both reachability sets and 1-step reachability sets for PAD and wPAD systems. Computing reachability sets is difficult for PRS in general. One of the reasons is that already the reachability sets of Petri nets are not semilinear. In [3] we show that the reachability sets of a given canonic PRS system R can be effectively computed provided the underlying multiset rewrite system $R_{||}$ is effectively semilinear. This is, for example, the case of canonic PAD systems due to the result of [7] concerning context-free multiset rewrite systems (BPP processes).

Theorem 5.1 ([3]) *Let $\mathcal{A}$ be a CH-automaton recognizing a set of process terms and R be a canonic PAD. Then the sets $Post^*_R(L(\mathcal{A}))$ and $Pre^*_R(L(\mathcal{A}))$ are computable and effectively representable by CH-automata.*

Using this theorem and the results of the previous section, we get the following.

Theorem 5.2 *For every PAD R and every CH-automaton $\mathcal{A}$, the sets $Post_R(L(\mathcal{A}))$, $Pre_R(L(\mathcal{A}))$, $Post^*_R(L(\mathcal{A}))$, and $Pre^*_R(L(\mathcal{A}))$ are computable and effectively representable by CH-automata.*

Proof. Theorem 2.1 implies that for every PAD R and every set of terms L, there exists a canonic PAD R' and a finite term substitution h such that $Post^*_R(L) = h(Post^*_{R'}(h^{-1}(L)))$ and $Post_R(L) = h(Post_{R''}(h^{-1}(L)))$, where R'' is the set R' restricted to rules labelled with actions of $Act(R)$. Hence, CH-automata representing the sets $Post^*_R(L(\mathcal{A}))$ and $Post_R(L(\mathcal{A}))$ are constructible due to closure properties of CH-automata and Theorems 5.1 and 4.1. The proof for $Pre^*_R(L(\mathcal{A}))$ and $Pre_R(L(\mathcal{A}))$ is analogous. □

Now we show that the previous theorem holds for wPAD as well. Recall that states of wPAD are pairs pt of a control state p and a term t. The sets of such states can be represented by *CHA-mappings*.

Definition 5.3 Let R be a wPRS. A *CHA-mapping* Λ is a mapping assigning to each control state $p \in M(R)$ a CH-automaton $\Lambda(p)$. A *CHA-mapping* Λ represents the set of states $L(\Lambda) = \{pt \mid p \in M(R), t \in L(\Lambda(p))\}$.

Theorem 5.4 *For every wPAD R and every CHA-mapping Λ, the sets $Post_R(L(\Lambda))$, $Pre_R(L(\Lambda))$, $Post^*_R(L(\Lambda))$, and $Pre^*_R(L(\Lambda))$ are computable and effectively representable by CHA-mappings.*

Proof. Let R be a wPAD. For each pair of control states $p, q \in M(R)$ we set $R_{p,q} = \{t_1 \overset{a}{\hookrightarrow} t_2 \mid pt_1 \overset{a}{\hookrightarrow} qt_2 \text{ is a rule of } R\}$. Note that each $R_{p,q}$ is a PAD system.

CHA-mapping Λ_1 representing $Post_R(L(\Lambda))$ is defined as follows. For each $q \in M(R)$, $\Lambda_1(q)$ is an CH-automaton satisfying

$$L(\Lambda_1(q)) = \bigcup_{p \in M(R)} Post_{R_{p,q}}\big(L(\Lambda(p))\big).$$

CHA-mapping Λ_2 representing $Post^*_R(L(\Lambda))$ is defined inductively with respect to ordering $<$ on set $M(R)$ of control states. For every minimal element r of $M(R)$, $\Lambda_2(r)$ is a CH-automaton satisfying $L(\Lambda_2(r)) = Post^*_{R_{r,r}}\big(L(\Lambda(r))\big)$. For non-minimal element q of $M(R)$, $\Lambda_2(q)$ is a CH-automaton satisfying

$$L(\Lambda_2(q)) = Post^*_{R_{q,q}}\Big(L(\Lambda(q)) \cup \bigcup_{p<q} Post_{R_{p,q}}\big(L(\Lambda_2(p))\big)\Big).$$

CHA-mappings Λ_1, Λ_2 are constructible due to Theorem 5.2 and the fact that CH-automata are closed under union. The proof for $Pre_R(L(\Lambda))$ and $Pre^*_R(L(\Lambda))$ is analogous. □

As mentioned in [3], the generic algorithm presented there can employ known algorithms computing semilinear overapproximations of reachability sets for Petri nets in order to compute overapproximations of reachability sets for general canonic PRS systems. If we use this approximative algorithm for canonic PRS instead of exact algorithm for canonic PAD system in Theorems 5.2 and 5.4, we get an algorithm computing overapproximations of reachability sets for general wPRS systems. Note that 1-step reachability sets for wPRS systems can still be computed precisely as Theorems 5.2 and 5.4 hold even for (w)PRS if we restrict our attention only to 1-step reachability sets.

6 Model checking of wPAD against EF logic

This section presents a straightforward application of Theorem 5.4. We consider a variant of EF logic combining both action-based and state-based approaches. We show that the global model checking problem of wPAD systems against this logic is decidable.

Formulae of EF logic are defined as

$$\varphi ::= P \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle a \rangle \varphi \mid \mathsf{EF}\varphi,$$

where P ranges over set AP of *atomic propositions* and a ranges over Act. Here, formulae are interpreted over states of wPRS systems. For each atomic proposition P, let $V(P)$ denotes its valuation, i.e. the set of states where P holds. We define when a state pt of a given wPRS system R *satisfies* a formula φ, written $R, pt \models \varphi$, by induction on the structure of φ.

$$
\begin{array}{lll}
R, pt \models P & \text{iff} & pt \in V(P) \\
R, pt \models \neg\varphi & \text{iff} & R, pt \not\models \varphi \\
R, pt \models \varphi_1 \wedge \varphi_2 & \text{iff} & R, pt \models \varphi_1 \text{ and } R, pt \models \varphi_2 \\
R, pt \models \langle a \rangle \varphi & \text{iff} & \exists qt' \text{ such that } pt \xrightarrow{a}_R qt' \text{ and } R, qt' \models \varphi \\
R, pt \models \mathsf{EF}\varphi & \text{iff} & \exists qt' \text{ such that } pt \xrightarrow{*}_R qt' \text{ and } R, qt' \models \varphi
\end{array}
$$

Theorem 6.1 *For every wPAD system R and every EF formula φ over atomic propositions with valuations given by CHA-mappings, the set of states of R satisfying φ is computable and effectively representable by a CHA-mapping.*

Proof. The theorem follows directly from Theorem 5.4 and closure properties of CH-automata. Here we mention just the induction step corresponding to operator $\langle a \rangle$. Let $\varphi = \langle a \rangle \psi$ and let CHA-mapping Λ recognizes all states satisfying ψ. We construct a CHA-mapping Λ', which recognizes all states where φ holds, to satisfy $L(\Lambda') = Pre_{R_a}\big(L(\Lambda)\big)$, where R_a is the set R restricted to rules with label a. Such a CHA-mapping Λ' is constructible due to Theorem 5.4. □

This theorem gives a positive answer to open questions formulated in [15], namely whether model checking of wBPP, wPA, and wPAD systems against action-based EF logic is decidable. Our result is tight as model checking of state extended PAD (defined as wPAD where rules may not respect the ordering on control states) against EF logic is already undecidable. In fact, the problem is undecidable even for the subclass of state extended PAD called *multiset automata* and EF formulae with the only atomic proposition *true* (this can be proved by the arguments of [7] showing that model checking of Petri nets against EF logic is undecidable).

7 Conclusion

We have presented an automata-based symbolic reachability analysis algorithm for the class of wPAD systems. This algorithm is based on the use of a class of unranked tree automata (called CH-automata) which can recognize sets of configurations closed under the algebraic properties of the sequential and parallel composition. We used the reachability analysis algorithm, together with one-step successor computation (and boolean operations on CH-automata), in order to define an algorithm for the global model checking of wPAD against the EF logic with regular atomic predicates. These results generalize those proved in [3] concerning the class of (canonic) PAD systems, which is a strict subclass of wPAD, pushing the known decidability limit of EF model checking further up in the (se/w)PRS hierarchy, and answering open questions left in [15].

As shown in [1], our symbolic reachability algorithm for wPAD can be used to compute under approximations of the set of reachable configurations of synchronized PAD (SPAD), a (Turing) powerful model introduced in [22] for modeling multithreaded programs (with dynamic creation of communicating processes and procedure calls).

References

[1] Bouajjani, A., J. Strejček and T. Touili, *On symbolic verification of weakly extended PAD*, Technical Report 2006-001, LIAFA, CNRS and University of Paris 7 (2006), full version of this paper.

[2] Bouajjani, A. and T. Touili, *Reachability Analysis of Process Rewrite Systems*, in: *Proc. of FSTTCS 2003*, LNCS **2914** (2003), pp. 74–87.

[3] Bouajjani, A. and T. Touili, *On computing reachability sets of process rewrite systems*, in: *Proceedings of RTA 2005*, LNCS **3467** (2005), pp. 484–499.

[4] Bruggemann-Klein, A., M. Murata and D. Wood, *Regular tree and regular hedge languages over unranked alphabets*, Research report (2001).

[5] Colcombet, T., *Rewriting in the partial algebra of typed terms modulo ac*, in: *Proceedings of INFINITY'02*, ENTCS **68** (2002).

[6] Comon, H., M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison and M. Tommasi, *Tree automata techniques and applications*, Available on: http://www.grappa.univ-lille3.fr/tata (1997).

[7] Esparza, J., *Decidability of model checking for infinite-state concurrent systems*, Acta Informatica **34** (1997), pp. 85–107.

[8] Esparza, J., *Grammars as processes*, in: *Formal and Natural Computing*, LNCS **2300** (2002).

[9] Esparza, J. and J. Knoop, *An automata-theoretic approach to interprocedural dataflow analysis*, in: *Proceedings of FOSSACS'99*, LNCS **1578**, 1999, pp. 14–30.

[10] Esparza, J. and A. Podelski, *Efficient algorithms for pre* and post* on interprocedural parallel flow graphs*, in: *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POLP'00)* (2000), pp. 1–11.

[11] Harrison, M. A., "Introduction to Formal Language Theory," Addison-Wesley, 1978.

[12] Jančar, P., A. Kučera and R. Mayr, *Deciding bisimulation-like equivalences with finite-state processes*, Theor. Comput. Sci. **258** (2001), pp. 409–433.

[13] Křetínský, M., V. Řehák and J. Strejček, *Extended process rewrite systems: Expressiveness and reachability*, in: *Proceedings of CONCUR'04*, LNCS **3170** (2004), pp. 355–370.

[14] Křetínský, M., V. Řehák and J. Strejček, *On extensions of process rewrite systems: Rewrite systems with weak finite-state unit*, in: *Proceedings of INFINITY'03*, ENTCS **98** (2004), pp. 75–88.

[15] Křetínský, M., V. Řehák and J. Strejček, *Reachability of Hennessy-Milner properties for weakly extended PRS*, in: *Proceedings of FSTTCS 2005*, LNCS **3821** (2005), pp. 213–224.

[16] Lugiez, D., *Counting and equality constraints for multitree automata*, in: *Proceedings of FoSSaCS 2003*, LNCS **2620** (2003), pp. 328–342.

[17] Lugiez, D. and P. Schnoebelen, *The regular viewpoint on PA-processes*, in: *Proc. of CONCUR'98*, LNCS **1466** (1998), pp. 50–66.

[18] Mayr, R., *Process rewrite systems*, Information and Computation **156** (2000), pp. 264–286.

[19] Qadeer, S. and J. Rehof, *Context-bounded model checking of concurrent software*, in: *Proceedings of TACAS'2005*, LNCS **3440** (2005), pp. 93–107.

[20] Seidl, H., T. Schwentick and A. Muscholl, *Numerical document queries*, in: *Proceedings of PODS'03* (2003), pp. 155–166.

[21] Touili, T., "Analyse symbolique de systèmes infinis basée sur les automates: Application à la vérification de systèmes paramétrés et dynamiques," Ph.D. thesis, University of Paris 7 (2003).

[22] Touili, T., *Dealing with communication for dynamic multithreaded recursive programs*, in: *Proceedings of VISSAS'05*, 2005.

Local bigraphs and confluence: two conjectures

(extended abstract)

Robin Milner

University of Cambridge

Bigraphs have been used to present a variety of models of concurrency within a single framework, which also provides a theory applicable to all the models. As we seek informatic understanding of extensive real-life systems that reconfigure themselves, we cannot expect that our present repertoire of abstract process calculi (including Petri nets, mobile ambients, CSP and π-calculus) will suffice. So, as we enlarge our repertoire of calculi —perhaps specific to a certain application (e.g. in biology or in pervasive computing)— there is a need for unifying theory.

The bigraphical model is an experiment in this direction. It is not a specific calculus, but rather a framework for defining and combining such calculi. To define a specific bigraphical reactive system (BRS) two ingredients are needed: its *signature* defines its *controls* (the kinds of nodes allowed), and its *reaction rules* define how bigraphs can reconfigure themselves.

Already the model has yielded some elements of a theory, especially of labelled transitions and behavioural congruences [6,4,5,7], which is applicable to a variety of BRSs. The present exercise addresses a different topic. First, in *local bigraphs* [8] we introduce a new treatment of names that allows them to have multiple *locality* (an example follows shortly). Similar work in bigraphs is by Bundgaard and Hildebrandt [3]. Second, we study the notion of *confluence* —i.e. independence among actions— in this setting, in the belief that it will arise frequently in applications. One need only think of modelling behaviour within a building: activity at one end of the building is largely independent of activity at the other end.

This summary omits some details, but should be accessible to those unfamiliar with bigraphs. It summarises work whose aims are as follows: to understand how activities in local bigraphs can conflict with one another, leading to non-confluence; to represent the λ-calculus —the classic setting for confluence studies— within local bigraphs; and thereby to learn conditions under which confluence can be assured within this wider setting. The work is in progress; the summary ends with two

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

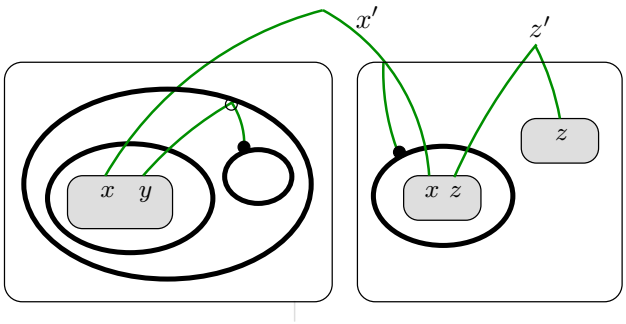


Fig. 1. A local bigraph $G : \langle\{xy\}, \{xz\}, \{z\}\rangle \to \langle\{x'\}, \{x'z'\}\rangle$

conjectures.

Mathematical framework: We work in s-categories. They differ from categories in that each arrow f has a *support* $|f|$, a finite set; composition $g \circ f$ is defined only if $|g| \cap |f| = \emptyset$, and then $|g \circ f| = |g| \cup |f|$. Two arrows f and g are *support equivalent*, $f \simeq g$, if they differ only by a bijection between their supports. Support is important for the notion of *occurrence* of one bigraph in another. For example, our Conjecture 1 rests upon analysis of when and how two redex occurrences can overlap each other.

1 Local bigraphs

Local bigraphs are arrows in an s-category whose objects are *interfaces*. An interface $I = \boldsymbol{X} = \langle X_0, \ldots, X_{m-1}\rangle$ has *width* m, a finite ordinal, and assigns to each *location* $i \in m$ a finite set X_i of *names*. The X_i need not be disjoint; thus, for example, any $x \in X_0 \cap X_1$ has dual locality.

If $J = \boldsymbol{Y}$ is another interface with width n, then a local bigraph $G : I \to J$ has m *sites* and n *roots* (or *regions*). Each region contains an unordered tree, whose root is the region and whose other members are either *nodes* or sites; the latter must be leaves. The interfaces dictate an assignment of names to each site and each region; the *inner* and *outer* names of G are those of I and J respectively. The support $|G|$ of G is its set of nodes; we say that F and G *overlap* if their supports are not disjoint.

Figure 1 shows a local bigraph with three sites (shaded) and two roots; the trees are represented by nesting. Each node may have *ports*, the number depending on the node's *kind* or *control* (not shown). The set of ports and inner names is partitioned into *links*; a link is either *free* (an outer name) or *bound* by a *binding port*. The example has two free links, x' and z', and one link bound by a port on the largest node. Binding ports are shown as circles, free ports as bullets.

There is a scoping discipline: if a link is bound, then its inner names and ports must lie within the node that binds it; if a link is free, with outer name x, then x must be located in every region that contains any inner name or port of the link.

The *composition* of $G : I \to J$ with $F : H \to I$, written $G \circ F$, is easy to define graphically: insert the roots of F in the sites of G, joining links at like names and eliding the names. Observe that, via composition, nodes in different regions can become separated by arbitrarily many node boundaries —while still sharing links.

An *agent* $a : \epsilon \to I$ has no sites; ϵ is the trivial interface with width 0. We use

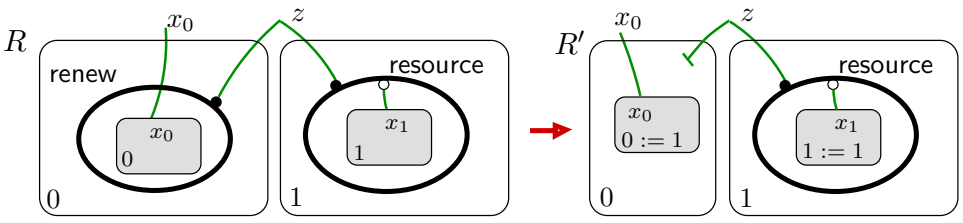


Fig. 2. A parametric reaction rule

lower-case letters for agents.

2 Reaction rules and λ-calculus

We are interested in *parametric (reaction) rules* that reconfigure agents. Such a rule has a redex $R : H \to K$ and a reactum $R' : H' \to K$, which may have different numbers m and m' of sites. A *parameter* for the rule is then an agent $a : H \oplus I$, with width m. The interface $H \oplus I$ has width m; it combines two interfaces H and I, each with width m, by taking the union of names at each location. H represents names of a to be bound by R; I represents names of a to be exported by extra free links through R.

Figure 2 shows a parametric rule where R and R' both have two sites. So it takes a parameter $a = a_0 \,\|\, a_1$ of width 2, with factors a_0 and a_1 each of unit width. The the parallel composition $\|$ is derivable from the tensor product in s-categories; if a and b have widths m and n and disjoint supports, then in $a \,\|\, b$ –with width $m{+}n$– they are placed side-by-side, sharing free links. Sites in R and R' are numbered; an assignment $j := i$ written in the j^{th} site of R' means that the reaction should place here a copy of the i^{th} factor of a. Thus the rule shown will discard a_0 and duplicate a_1, putting one copy at each site of R'. (We omit details of how each copy's names are determined.)

We can think of the rule as the renew node fetching from the resource node (via the shared link z) a new copy of its resource a_1. Since R has two regions, the renew and resource nodes may be arbitrarily far apart in a large bigraph containing an occurrence of the redex R; so the rule offers the possibility of action at a distance.

Let us now define a certain λ-calculus, Λ_{sub}, in the usual way. It is a version with explicit substitutions, but with coarser steps than that of Abadi et al [1]. The terms are

$$M \quad ::= \quad x \;\big|\; \lambda x M \;\big|\; MN \;\big|\; M\,[x{:=}N]$$

The final term construction should be read 'M where x means N'; it should not to be confused with $\{N/x\}M$, the result of *replacing* all free occurrences of x in M by N.

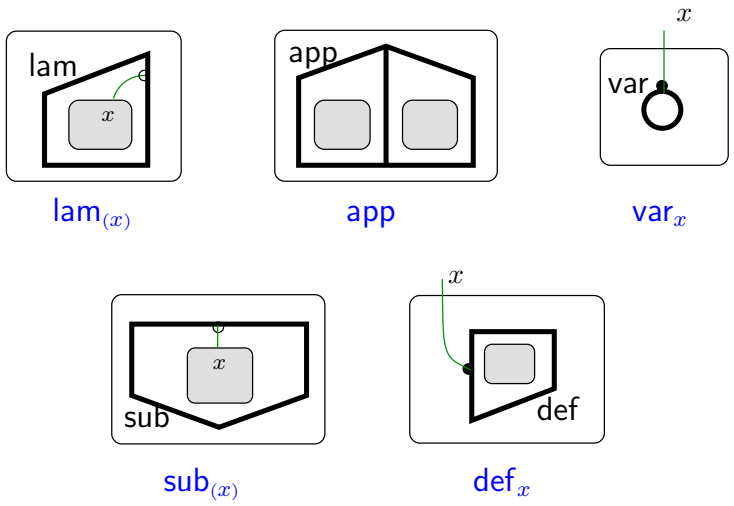


Fig. 3. Ions for the ˆΛBIG, with their algebraic representation

Definition 2.1 (reduction) The reduction rules in Λ_{sub} are as follows:

$$(\lambda x M)N \longrightarrow M\,[x{:=}N]$$

$$(\{x/y\}M)\,[x{:=}N] \longrightarrow (\{N/y\}M)\,[x{:=}N] \quad \text{where } M \text{ has a unique free occurrence of } y$$

$$M\,[x{:=}N] \longrightarrow M \quad \text{where } M \text{ has no free occurrence of } x\ .$$

Reductions may be applied to any subterm of a term. ∎

Thus reductions are allowed even inside an explicit substitution. In the second rule, $\{x/y\}M$ distinguishes a particular free occurrence of x to be replaced by N. The three rules together achieve β-reduction. The explicit substitution $[x{:=}N]$ acts 'at a distance' on each free occurrence of x in turn, rather than migrating a copy of itself towards each such occurrence as in [1].

We now turn to ˆΛBIG, the BRS corresponding to Λ_{sub}. Figure 3 shows its signature both graphically and algebraically. There are five controls (kinds of node), shown as *ions* (elementary bigraphs); a var-node has no sites, an app-node has two, and the rest have one. lam- and sub-nodes bind a link; var- and def-nodes have one port. The shapes of a node is unimportant, except that the shape of the app-node signifies that its sites are in left-to-right order. [1] Note that binding names are parenthesized.

To export free names from their occupants, the ions with sites are generalised to

$$\mathsf{lam}_{(x)} \oplus \mathsf{id}_Z \quad \mathsf{app} \oplus (\mathsf{id}_Y \mid \mathsf{id}_Z)$$

$$\mathsf{sub}_{(x)} \oplus \mathsf{id}_Z \quad \mathsf{def}_x \oplus \mathsf{id}_Z\ .$$

The app-ion exports names Y from its first site, Z from its second.

[1] Multiple-site Controls are definable from single-site ones, site, with the help of a sorting discipline.

Two new operators appear here. In this abstract we do not define operators formally, but illustrate their meaning by examples. A *prime* composition $F \,|\, G$ is like $F \parallel G$ (and derivable from it), but it merges the outer regions of F and G into one. The operator $\oplus$ is called *extension*. The extension $I \oplus I'$ of interfaces (with same width) was defined earlier. Given $G : I \to J$ and $\omega : I' \to J'$ (a wiring, i.e. a node-free bigraph) one can form $G \oplus \omega : I \oplus I' \to J \oplus J'$ provided the interface extensions are defined; it has the same tree structure as G, but the linkage of G is extended by adding the linkage of ω. Thus $\mathsf{id}_Y \,|\, \mathsf{id}_Z$, with inner width 2 and outer width 1, is a suitable extension for app; it exports the union of the inner name-sets Y and Z as outer names. The operators $\circ$, $\parallel$, $|$ and $\oplus$, though partial, have a rich algebraic theory.

The free names in a bigraph built from the above ions correspond exactly to the free variables in a λ-term. Thus $\lambda x\, x(xy)$ will translate into the bigraph

$$(\mathsf{lam}_{(x)} \oplus \mathsf{id}_y) \circ (\mathsf{app} \oplus (\mathsf{id}_x \,|\, \mathsf{id}_{xy})) \circ (\mathsf{var}_x \parallel ((\mathsf{app} \oplus (\mathsf{id}_x \,|\, \mathsf{id}_y)) \circ (\mathsf{var}_x \parallel \mathsf{var}_y))) \ .$$

(Here a set such as $\{xy\}$ has been written without curly brackets.) Of course, this notation is not recommended for developing λ-calculus theory! – but it has the advantage that the free names exported with each term constructor are made explicit.

We now translate Λ_{sub} into $\hat{}\Lambda\textsc{big}$. The translation function $[\![M]\!]_X$ is indexed by the set X, which must include all the free variables of M. Thus each term M has many bigraph images. This technique was used to model the asynchronous π-calculus [4].

Definition 2.2 (λ-terms into bigraphs)

$$\begin{aligned}
[\![x]\!]_{X \uplus x} &\stackrel{\text{def}}{=} \mathsf{var}_x \oplus X \\
[\![\lambda x M]\!]_X &\stackrel{\text{def}}{=} (\mathsf{lam}_{(x)} \oplus \mathsf{id}_X) \circ [\![M]\!]_{X \uplus x} \\
[\![MN]\!]_X &\stackrel{\text{def}}{=} (\mathsf{app} \oplus (\mathsf{id}_X \,|\, \mathsf{id}_X)) \circ ([\![M]\!]_X \parallel [\![N]\!]_X) \\
[\![M[x{:=}N]]\!]_X &\stackrel{\text{def}}{=} (\mathsf{sub}_{(x)} \oplus \mathsf{id}_X) \circ ([\![M]\!]_{X \uplus x} \,|\, ((\mathsf{def}_x \oplus \mathsf{id}_X) \circ [\![N]\!]_X)) \ .
\end{aligned}$$

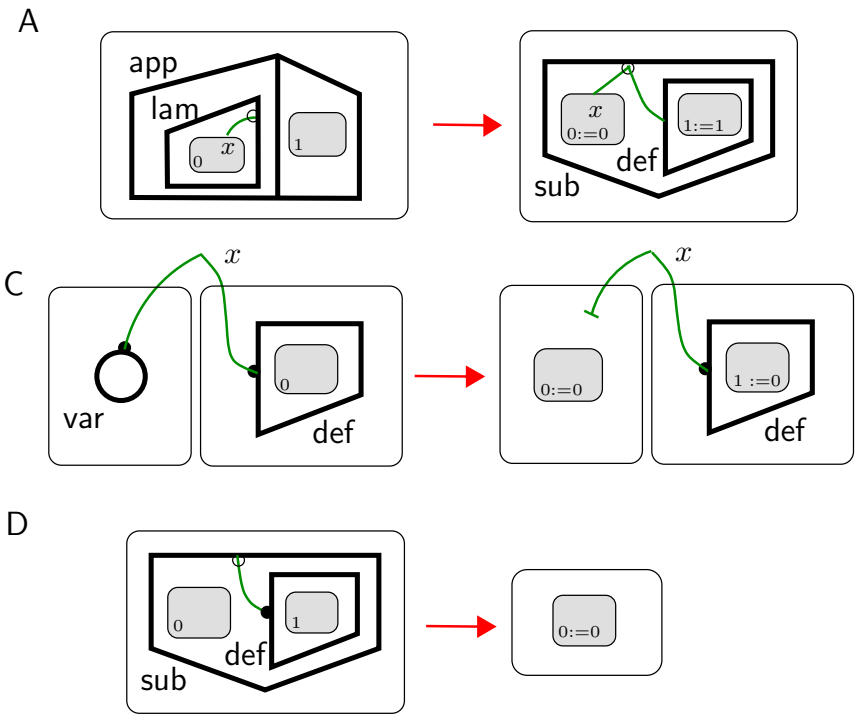
■

We shall not discuss this translation fully. But it is worth noting that alpha-convertible λ-terms have equal images; this is because bound names are elided by composition. We are now ready to present the reaction rules for the BRS $\hat{}\Lambda\textsc{big}$.

Definition 2.3 (dynamics) $\hat{}\Lambda\textsc{big}$ has three reaction rules: A (apply), C (copy) and D (discard). They are shown both graphically and algebraically in Figure 4. ■

Note that rule C has width 2. Thus, in C, an occurrence of the 'variable' x may be distant from the defining equation that will replace it with a 'term'. This rule exploits the multiple locality of names in local bigraphs; it is similar to the rule of Figure 2.

We now assert that reaction in $\hat{}\Lambda\textsc{big}$ exactly matches reduction in Λ_{sub}:



	R	R'
A	$\mathsf{app} \circ (\mathsf{lam}_{(x)} \parallel \mathsf{id})$	$\mathsf{sub}_{(x)} \circ (\mathsf{id}_x \mid \mathsf{def}_x)$
C	$\mathsf{var}_x \parallel \mathsf{def}_x$	$\mathsf{id} \parallel \mathsf{def}_x$
D	$\mathsf{sub}_{(x)} \circ (\mathsf{id} \mid \mathsf{def}_x)$	id

Fig. 4. Parametric reaction rules for ˆΛBIG

Proposition 2.4 (reaction matches reduction) *$[\![M]\!]_X \longrightarrow\!\!\triangleright\, g$ if and only if $M \longrightarrow\!\!\triangleright\, M'$ for some M' such that $[\![M']\!]_X \eqsim g$.*

In fact, for each reduction by a rule for Λ_{sub} there is a matching reaction by the corresponding rule for ˆΛBIG, and conversely. In a recent draft O'Conchuir [9] has proved (strong) confluence directly for Λ_{sub}, so this translates immediately into a confluence proof for ˆΛBIG. Our purpose here is different; we use the bigraphical representation to illustrate the confluence properties that we seek for bigraphs in general.

3 Confluence in bigraphs

Recall that for any given reduction or reaction relation '$\longrightarrow\!\!\triangleright$' there are three familiar notions of *confluence*, shown in Figure 5. They all say that if g can react to become either g_0 or g_1, then these two reacta can in turn react to reach a common result. Clearly *one-step* $\Rightarrow$ *strong* $\Rightarrow$ *weak*, and it is well-known that these implications are strict in general. The most positive result for a BRS would be that strong confluence holds outright. This is indeed true (the Church-Rosser theorem) for the classical λ-calculus, and (by O'Conchuir) for Λ_{sub} also. However, in bigraphs we cannot expect

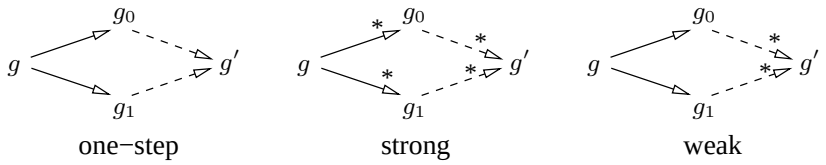


Fig. 5. Three notions of confluence

this in general. Instead, we shall look for conditions that ensure non-interference between two competing reactions $g \longrightarrow\!\triangleright\, g_0$ and $g \longrightarrow\!\triangleright\, g_1$; such conditions may depend on the reaction rules that underlie the two translations, and on the extent to which the two redices overlap (if at all) in g. Moreover, it is in general easier to establish weak confluence in such cases.

If we succeed in showing that weak confluence always holds for a certain class of agents under certain reaction rules, and if this class is itself preserved by reaction, then we may look to well-known methods from the theory of the λ-calculus that allow us to deduce strong from weak confluence. One such method is based upon *developments* [2]. A development is a reduction sequence $M \longrightarrow\!\triangleright\, M_1 \longrightarrow\!\triangleright\, M_2 \longrightarrow\!\triangleright\, \cdots$ in which the only redices reduced are the residuals of an arbitrary set of redices present initially in M. The method is based upon the theorem that if all developments are of finite length, then weak confluence implies strong confluence.

Before going further, we note that BRSs can be wilder than the λ-calculus! One property, used again and again in case analyses for the λ-calculus, is that when a term contains two redices then they are either disjoint or else nested (one inside the other). This fails for ground redices in BRSs; worse, it even fails for the parametric redices underlying them. Indeed, Figure 6 shows two possible parametric redices which are intimately entwined, each partly inside the other.

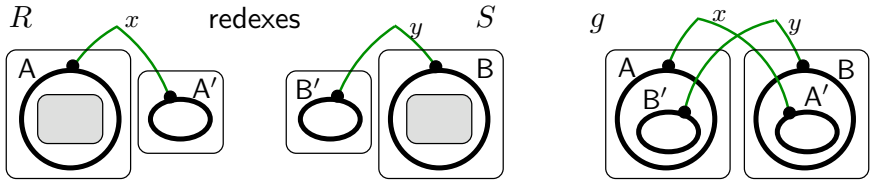


Fig. 6. An agent g containing two intertwined redices R and S

We do not know whether this property —redices nested or disjoint— is essential for weak confluence, or for finiteness of developments, in a BRS. However, recent investigation has explored a classification of ways in which two competing redices can overlap. If a parametric redex R supports a reaction $g \longrightarrow\!\triangleright\, g_0$, then $r = (R \oplus \omega) \circ a$ occurs in g, for some parameter a and wiring ω. Similarly, if redex S supports a reaction $g \longrightarrow\!\triangleright\, g_1$, then $s = (S \oplus \zeta) \circ b$ occurs in g. The *ground* redices r and s can overlap in different ways; for example s may not overlap with R, but may partly overlap with a. The investigation identifies four principal cases for such overlap, and claims that under certain further conditions the weak confluence diagram can be completed by $g_0 \longrightarrow\!\triangleright^* g'$ and $g_0 \longrightarrow\!\triangleright^* g'$. As this work is not complete we confine ourselves at present to two indeterminate conjectures about reactions in BRSs:

Conjecture 1 (weak confluence in $\hat{}\Lambda$BIG) *Weak confluence holds for certain sets of agents in certain BRSs, including the set of all images of Λ_{sub}-terms in*

$\hat{}\Lambda$Big.

Conjecture 2 (finite developments in $\hat{}\Lambda$Big) *Developments are finite for certain sets of agents in certain BRSs.*

Together, these two results will lead to strong confluence for the agents mentioned.

Conjecture 1 is reasonably firm, since (as indicated) much of the analysis has been done. Conjecture 2 is left vague at present. It is possible that, in $\hat{}\Lambda$Big, developments are finite only under some constraint. O'Conchuir's detailed study [9] may help to identify such a constraint.

Conclusion

The aim of this work is not to find yet another proof of the Church–Rosser theorem for a variant of the λ-calculus, but rather to learn from such proof techniques in order to analyse confluence for a wider class of agents and reaction rules than that for which it has hitherto been studied. This will lead to a better understanding not only of practically useful BRSs, but also of confluence itself.

References

[1] Abadi, M., Cardelli, L., Curien, P-L. and Levy, J-J. (1991), Explicit substitutions. Journal of Functional Programming 1, pp375–416.

[2] Barendregt, H. (1984), The Lambda Calculus: its Syntax and Semantics. North Holland.

[3] Bundgaard, M. and Hildebrandt, T. (2006), Bigraphical semantics of higher-order mobile embedded resources with local names. Proc. GT-VC 2005, Electronic Notes in Theoretical Computer Science 154(2), pp7–29.

[4] Jensen, O-H. and Milner, R. (2003), Bigraphs and transitions. Proc 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), 2003, 16pp.

[5] Jensen, O.H. and Milner, R. (2004), Bigraphs and mobile processes (revised). Technical Report 580, University of Cambridge Computer Laboratory. Available from http://www.cl.cam.ac.uk/users/rm135.

[6] Leifer, J. and Milner, R. (2000), Bisimulation congruences for reactive systems. Proc. CONCUR2000, LNCS, Vol 1877, pp243–258.

[7] Leifer, J. and Milner, R. (2006), Link graphs, transitions and Petri nets. To appear in Mathematical Structures in Computer Science.

[8] Milner, R. (2004), Bigraphs whose names have multiple locality. Technical Report UCAM-CL-TR-603, University of Cambridge, Computer Laboratory. Available from http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-603.pdf .

[9] O'Conchuir, S. (2006), Λ_{sub} as an explicit substitution calculus (draft).

General reversibility

Vincent Danos

CNRS & Université Paris 7

Jean Krivine

INRIA Rocquencourt & Université Paris 6

Paweł Sobociński[1]

Computer Laboratory, University of Cambridge

Abstract

The first and the second author introduced reversible CCS (RCCS) in order to model concurrent computations where certain actions are allowed to be reversed. Here we show that the core of the construction can be analysed at an abstract level, yielding a theorem of pure category theory which underlies the previous results. This opens the way to several new examples; in particular we demonstrate an application to Petri nets.

1 Introduction

The reversible calculus of communicating systems (RCCS) [1] is essentially Milner's CCS [9] with the caveat that some observable actions in the standard labelled transition system (LTS) semantics are understood to be reversible. Technically, the theoretical development involved the engineering of explicit syntax for keeping track of a computation history. Such a history, together with a CCS term, forms the configuration of a given process. Appropriate new structural operational semantics (SOS) rules allowed the reversible components of a given state's history to be undone. Phillips and Ulidowski [10] proposed a different approach to keeping the record of a computation's history; instead of keeping an explicit representation of history together with an unevaluated term, they keep the structure of terms essentially unaltered by making the SOS rules *static*. Causality is kept track of by tagging actions with so-called communication keys.

In [2], it was argued that a calculus such as RCCS (or CCSK of [10]) is suited for modelling *transactions* – ie computations where several agents interact in order

[1] Research partially supported by EPSRC grant EP/D066565/1.

to agree on a common irreversible action; see [3] for example. Indeed, it seems that guaranteeing the soundness of such transactions is easy enough since policies are normally specified by requiring the local states of the participants to satisfy certain criteria. On the other hand, completeness seems to be more difficult, since the existence of a possible computation leading to all of the agents having the required state does not guarantee that such a state will be reached – for instance, the agents may deadlock while racing to obtain the necessary shared resources. If we stipulate that the actions leading to transactions are reversible and enrich the participants with histories, meaning that the intermediate actions can be undone, the irreversible computations are "essentially" the transactions. More concretely, it was shown in [2] that the LTS where the labels are taken to be the transactions and the LTS of processes with histories and reversible actions, where the reversible actions are equated with τs, are weakly bisimilar.

In this paper we show that the design of a calculus such as RCCS involves an underlying abstract construction of the history category from a category of computations. The fact that the computations agree essentially with the causal (irreversible) computations in the original category is captured by an equivalence of categories.

The main contributions of this paper are:

(i) the observation that subcategories $\mathcal{R}$ of reversible and $\mathcal{I}$ of causal computations form a factorisation system $\langle \mathcal{I}, \mathcal{R} \rangle$ on the category of computations $\mathbf{C}$ (cf §3);

(ii) given a factorisation system $\langle \mathcal{I}, \mathcal{R} \rangle$ on $\mathbf{C}$, an explicit construction of the "category of histories" $h_\star(\mathbf{C}, \mathcal{R})$ (cf Definition 4.3);

(iii) a proof that $h_\star(\mathbf{C}, \mathcal{R})$ essentially follows from a free construction; concretely we prove that $h_\star(\mathbf{C}, \mathcal{R})$ is equivalent to a certain category of fractions (cf Theorem 4.5);

(iv) an equivalence of categories $h_\star(\mathbf{C}, \mathcal{R}) \simeq \mathcal{I}$ (cf Theorem 4.4) – this is the main result of the paper and guarantees that in order to capture the causal computations it is enough to keep the reversible parts of a computation along as part of the state and allow these histories to be undone;

(v) a direct application of Theorem 4.4 to the categories of computations induced by Petri nets;

(vi) an explanation of how Theorem 4.4 relates to the previous work [2] concerning RCCS. In particular, a weak bisimulation that relates the LTS of transactions to the LTS of reversible histories where the reversible actions are treated as internal (cf Theorem 5.3).

Structure of the paper

In §2 we recall the basic concepts of categories of fractions and factorisation systems. In §3 we introduce several examples, including Petri nets, and show that the sets of causal and reversible computations form factorisation systems. The construction of the history category together with our main Theorem 4.4 is given in §4. Finally, in §5 we explore the connections with labelled transition systems. The paper assumes a basic acquaintance with the categorical notions of adjunctions and

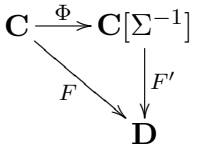
symmetric monoidal (sm) categories.

2 Categories of fractions and factorisation systems

Categories of fractions

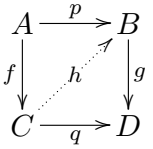
Given a category $\mathbf{C}$ and an arbitrary class of morphisms Σ, we denote by $\mathbf{C}[\Sigma^{-1}]$ the *category of fractions* obtained by "freely" adding formal inverses to the arrows of Σ (see, for instance [5]).

The category of fractions is characterised by a universal property: the existence of a functor $\Phi\colon \mathbf{C} \to \mathbf{C}[\Sigma^{-1}]$ which sends each arrow in Σ to an isomorphism, and moreover, given a category $\mathbf{D}$ and a functor $F\colon \mathbf{C} \to \mathbf{D}$ which takes each arrow in Σ to an isomorphism, the existence of a unique functor $F'\colon \mathbf{C}[\Sigma^{-1}] \to \mathbf{D}$ such that $F'\Phi = F$.



Factorisation systems

Given a category $\mathbf{C}$ and two arrows $f, g \in \mathbf{C}$ we shall write $f \perp g$ if f and g satisfy the following property: given a commutative diagram with p, q arbitrary morphisms of $\mathbf{C}$ there exists a unique morphism $h\colon C \to B$ such that $gh = q$ and $hf = p$, as illustrated. Notice that $\perp$ is not symmetric. Given an arbitrary set $\mathcal{X}$ of arrows of $\mathbf{C}$ there are two closure operations which use $\perp$:



$$\mathcal{X}^{\perp} = \{\, y \text{ in } \mathbf{C} \mid \forall x \in \mathcal{X}.\, x \perp y \,\} \text{ and } \mathcal{X}^{\top} = \{\, y \text{ in } \mathbf{C} \mid \forall x \in \mathcal{X}.\, y \perp x \,\}.$$

If we let $\mathrm{Iso}(\mathbf{C})$ ($\mathrm{Ar}(\mathbf{C})$) be the class of all isomorphisms (morphisms) of $\mathbf{C}$ then it's immediate that $\mathrm{Iso}(\mathbf{C})^{\perp} = \mathrm{Ar}(\mathbf{C}) = \mathrm{Iso}(\mathbf{C})^{\top}$.

The following are standard properties enjoyed by the closure operations:

Proposition 2.1

(i) $\mathcal{X}^{\perp\top\perp} = \mathcal{X}^{\perp}$;
(ii) $\mathcal{X}^{\top\perp\top} = \mathcal{X}^{\top}$;
(iii) $\mathcal{X} \subseteq \mathcal{X}' \Rightarrow \mathcal{X}'^{\perp} \subseteq \mathcal{X}^{\perp}$
(iv) $\mathcal{X} \subseteq \mathcal{X}' \Rightarrow \mathcal{X}'^{\top} \subseteq \mathcal{X}^{\top}$.

Following [4], we define a *prefactorisation system* as follows:

Definition 2.2 [Prefactorisation system] A prefactorisation system for a category $\mathbf{C}$ consists of two classes $\mathcal{I}$, $\mathcal{R}$ of arrows of $\mathbf{C}$ such that $\mathcal{I}^{\perp} = \mathcal{R}$ and $\mathcal{R}^{\top} = \mathcal{I}$.

By the first two parts of Proposition 2.1 it is immediate that for any class of arrows $\mathcal{X}$ of $\mathbf{C}$, $\langle \mathcal{X}^{\top}, \mathcal{X}^{\top\perp} \rangle$ and $\langle \mathcal{X}^{\perp\top}, \mathcal{X}^{\perp} \rangle$ are prefactorisation systems.

The following are some of the well-known properties of prefactorisation systems [4]:

Proposition 2.3 *Suppose that $\langle \mathcal{I}, \mathcal{R} \rangle$ is a prefactorisation system on $\mathbf{C}$. Then:*

(i) $\mathrm{Iso}(\mathbf{C}) \subseteq \mathcal{I}$, $\mathrm{Iso}(\mathbf{C}) \subseteq \mathcal{R}$ *and* $\mathcal{I} \cap \mathcal{R} = \mathrm{Iso}(\mathbf{C})$*;*

(ii) $\mathcal{I}$ *and* $\mathcal{R}$ *are closed under composition.*

The conclusion of Proposition 2.3 implies that $\mathcal{I}$ and $\mathcal{R}$ are actually subcategories of $\mathbf{C}$ since they contain the identities and are closed under composition. We shall take advantage of this by often confusing $\mathcal{I}$ and $\mathcal{R}$ with the subcategories they form the arrows of.

Definition 2.4 [Factorisation system] A prefactorisation system $\langle \mathcal{I}, \mathcal{R} \rangle$ on $\mathbf{C}$ is a factorisation system if every arrow p in $\mathbf{C}$ can be written $p = g \circ f$ for some f in $\mathcal{I}$ and g in $\mathcal{R}$.

Example 2.5 Clearly $\langle \mathbf{C}, \mathrm{Iso}(\mathbf{C}) \rangle$ and $\langle \mathrm{Iso}(\mathbf{C}), \mathbf{C} \rangle$ are factorisation systems in any category. Probably the most well-known factorisation system is of course $\langle \mathcal{E}, \mathcal{M} \rangle$ in the category of sets **Set**, where $\mathcal{E}$ is the class of surjections and $\mathcal{M}$ is the class of injections.

The following is a well-known property of factorisation systems:

Lemma 2.6 $\langle \mathcal{I}, \mathcal{R} \rangle$*-factorisation is unique up to isomorphism: if* $p \colon A \to B$ *in* $\mathbf{C}$ *can be factorised* $p = g_1 f_1$ *and also* $p = g_2 f_2$ *where* $f_i \colon A \to C_i$ *is in* $\mathcal{I}$ *and* $g_i \colon C_i \to B$ *is in* $\mathcal{R}$ *for* $i = 1, 2$*, then there exists a unique isomorphism* $h \colon C_1 \to C_2$ *such that* $h f_1 = f_2$ *and* $g_2 h = g_1$

3 Reversibility

Following the theoretical exposition, we give a number of motivating examples of factorisation systems. We shall consider categories of computations which decompose into an underlying set of atomic actions, some of which are a priori specified as reversible. Given a computation which consists of both types of actions, it should be possible to break it up into a *causal* (non-reversible) component followed by a maximal *reversible* component. If we denote the causal computations by $\mathcal{I}$ and the reversible computations by $\mathcal{R}$, it turns out that $\langle \mathcal{I}, \mathcal{R} \rangle$ usually forms a factorisation system on the category of computations.

Example 3.1 [Single-threaded reversibility] Consider an alphabet $\Sigma = I + R$ for some sets I and R; we think of I as a set of irreversible atomic actions and R as a set of reversible atomic actions. Let Σ^* denote the free monoid over Σ considered as a one-object category.

Let $\mathcal{R} = R^*$ and let $\mathcal{I} = \mathcal{R}^\top = \Sigma^* I + \epsilon$ – the set of all strings which end with an irreversible action, together with the empty string. Then $\langle \mathcal{I}, \mathcal{R} \rangle$ is a factorisation system on Σ^*.

Example 3.2 [Multi-threaded reversibility] Let $\mathbf{C}$ be the free SM category on a graph G – ie one first forms the free category on G and then the free SM category on the resulting category. We think of the vertices of G as representing the states of a particular thread of computation, and the edges as possible actions. Then, following this intuition, the arrows of $\mathbf{C}$ represent multithreaded computations of finitely many non-communicating processes, with the tensor product $\otimes$ representing parallel composition.

Suppose that the edges of G are partitioned into two sets, I and R. Let G_R denote the graph with the same nodes as G but with the edges restricted to the members of R.

Let $\mathcal{R}$ be the free SM category on G_R. Clearly $\mathcal{R}$ is a subcategory of $\mathbf{C}$ in a canonical way. Let $\mathcal{I} = \mathcal{R}^\top$. It is easy to verify that $\mathcal{I}$ is the smallest subcategory of $\mathbf{C}$ which contains the isomorphisms of $\mathbf{C}$, arrows of the form $i\alpha$ with $i \in I$ and whose arrows are closed under $\otimes$. Then $\langle \mathcal{I}, \mathcal{R} \rangle$ is a factorisation system on $\mathbf{C}$.

It is instructive to consider a more substantial example in order to illustrate the theory. Here we shall consider Petri nets as SM categories in the tradition of [8]. Note, however, that we do not deal with *strict* symmetric monoidal categories. We shall first need to recall the notion of a *tensor scheme* [6] and the associated notion of a free SM category on a tensor scheme; indeed, as we shall see, tensor schemes are very closely related to Petri nets. Note that tensor schemes can also be used in order to construct ordinary (ie non-symmetric) free monoidal categories.

Definition 3.3 [Tensor scheme]A tensor scheme $\mathcal{S}$ consists of a set V of vertices, a set E of edges, and functions $s, t : E \to V^*$, where V^* is the free monoid (the set of finite words) on V. Every tensor scheme leads to a free SM category $\mathbf{C}$ – see [6] for details. Intuitively, the objects of $\mathbf{C}$ can be seen as finite words (ie the product in V^* is interpreted as $\otimes$ in $\mathbf{C}$) in V and the arrows of $\mathbf{C}$ are generated freely from the basic edges in E. Concretely, the arrows can be seen as certain equivalence classes or as certain string diagrams; see [11]. Notice that the procedure described in Example 3.2 can be seen as a special case of a tensor scheme (where all the edges have one letter words as sources and targets).

Definition 3.4 [Petri net] A *Petri net* $\mathcal{N}$ with a set of states S and set of transitions T is a graph $s, t \colon T \to S^\oplus$ where $S^\oplus$ is the free commutative monoid on S. A *Petri category* $\mathbf{C}_\mathcal{N}$ is the free SM category on $\mathcal{N}$, considered as a tensor scheme. [2]

The Petri category $\mathbf{C}_\mathcal{N}$ can be thought of as the category with arrows the (truly) concurrent computations of a net $\mathcal{N}$.

Example 3.5 [Petri net reversibility] Suppose that the set T of transitions $\mathcal{N}$ can be partitioned $T = I + R$, where the set I contains the transitions which are deemed irreversible and R the transitions deemed reversible. We obtain a factorisation system $\langle \mathcal{I}, \mathcal{R} \rangle$ as in our previous examples.

Let $\mathcal{R}$ be the free SM category on $\mathcal{N}_R$, the Petri net with the same places as $\mathcal{N}$ and with R as its set of transitions, considered as a tensor scheme; it is clearly a subcategory of $\mathbf{C}_\mathcal{N}$ in a canonical way. Let $\mathcal{I} = \mathcal{R}^\top$ – the arrows of $\mathcal{I}$ can be described roughly as in Example 3.2. The pair $\langle \mathcal{I}, \mathcal{R} \rangle$ forms a factorisation system on $\mathbf{C}_\mathcal{N}$.

Consider the concrete example of a net illustrated in Figure 1, where precisely the unfilled transitions (g_1 and g_2) are taken to be reversible. Suppose that places x_1 and x_2 initially contain one token each; intuitively, we can consider places x_1 and x_2 as agents which each have an option of committing to two transactions:

[2] One fixes a particular ordering of places for the source and the target of each transition, the order is immaterial.

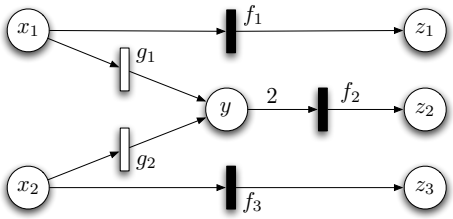


Fig. 1. A simple Petri net, the filled transitions are irreversible.

x_1 can commit to either f_1 or f_2 while x_2 can commit to f_2 or f_3. In terms of $\mathbf{C}_{\mathcal{N}}$ this amounts to the fact that there are arrows $f_1 \colon x_1 \to z_1$, $f_3 \colon x_2 \to z_3$ and $f_2.g_1 \otimes g_2 \colon x_1 \otimes x_2 \to z_2$. Notice that if x_1 chooses to perform g_1 and x_2 commits to f_3 then the computation begun by x_1 is stuck unless the g_1 transition can be reversed and f_1 chosen instead.

Consider the effect of adding new transitions $g_{1\star}$ and $g_{2\star}$ to act as the inverses of g_1 and g_2 respectively. If we deem that reversed computations are the same as doing nothing then the resulting Petri category is just $\mathbf{C}_{\mathcal{N}}[\mathcal{R}^{-1}]$. However, this setting is clearly unsuitable to model the expected behaviour of the net: consider starting with a single token in x_2 and performing the g_2 transition. Since now $g_{1\star}$ is enabled, we can perform $g_{1\star}$ and then f_1, thus arriving at a behaviour which is not in the specification – x_2 being able to commit to action f_1.

4 Histories

A key technical feature of RCCS is that histories are kept as part of the state, which allows reversible moves to be backtracked correctly. Here we repeat the construction at a higher level of abstraction, assuming only the presence of a factorisation system.

Definition 4.1 [Category $h(\mathbf{C}, \mathcal{R})$ of histories] Suppose that $\langle \mathcal{I}, \mathcal{R} \rangle$ is a factorisation system on $\mathbf{C}$. Let $h(\mathbf{C}, \mathcal{R})$ be the category with:

- objects: arrows g in $\mathcal{R}$;
- arrows: commutative diagrams, as illustrated, where f is in $\mathbf{C}$ and $f' \in \mathcal{I}$.

$$\begin{array}{ccc} P_1 & \xrightarrow{f'} & P_2 \\ {\scriptstyle g_1}\downarrow & & \downarrow{\scriptstyle g_2} \\ Q_1 & \xrightarrow[f]{} & Q_2 \end{array}$$

Notice that given an object $g_1 \colon P_1 \to Q_1$ in $h(\mathbf{C}, \mathcal{R})$ and an arbitrary arrow $f \colon Q_1 \to Q_2$, there exists a unique up-to-isomorphism object g_2 of $h(\mathbf{C}, \mathcal{R})$ and arrow $f' \colon P_1 \to P_2$ in $\mathcal{I}$ such that $\langle f, f' \rangle : g_1 \to g_2$ is in $h(\mathbf{C}, \mathcal{R})$ – here $g_2 \circ f'$ is just the $\langle \mathcal{I}, \mathcal{R} \rangle$-factorisation of $f \circ g_1$. Notice that if $f \in \mathcal{R}$, then using the fact that arrows of $\mathcal{R}$ compose and uniqueness of factorisation (Lemma 2.6), we have that $f' \in \mathrm{Iso}(\mathcal{C})$.

Recall from Proposition 2.3 that we can consider $\mathcal{I}$ to be a category. There is an obvious functor $M \colon h(\mathbf{C}, \mathcal{R}) \to \mathcal{I}$ which takes an object $g_1 \colon P_1 \to Q_1$ to P_1 and the diagram above to the arrow $f' \colon P_1 \to P_2$. Returning to our intuitions, this functor takes a computation to its causal (non-reversible) component. Using the final remark of the previous paragraph, M sends arrows which have a lower

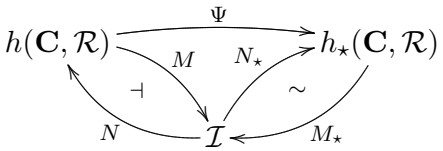


Fig. 2. Histories and causal computations.

component in $\mathcal{R}$ to isomorphisms.

There is also a (full and faithful) functor $N\colon \mathcal{I} \to h(\mathbf{C},\mathcal{R})$, which takes an object $P_1 \in \mathcal{I}$ to the identity on P_1 (null history) and a morphism $f\colon P_1 \to P_2$ to the illustrated diagram.

$$\begin{array}{ccc} P_1 & \xrightarrow{f} & P_2 \\ \downarrow & & \downarrow \\ P_1 & \xrightarrow[f]{} & P_2 \end{array}$$

Proposition 4.2 *N is left adjoint to M.*

Proof. Given $g_1\colon P_1 \to Q_1 \in h(\mathbf{C},\mathcal{R})$, consider the illustrated morphism $\epsilon_{g_1} = \langle g_1, \mathrm{id}\rangle : NM(g_1) \to g_1$. It is easy to verify that ϵ defines a natural transformation $NM \Rightarrow \mathrm{id}_{h(\mathbf{C},\mathcal{R})}$ – it is the counit of the adjunction. The unit is trivial as $MN = \mathrm{id}_{\mathcal{I}}$, and the triangle identities are easily checked. □

$$\begin{array}{ccc} P_1 & \longrightarrow & P_1 \\ \downarrow & & \downarrow{\scriptstyle g_1} \\ P_1 & \xrightarrow[g_1]{} & Q_1 \end{array}$$

Recall that our intuition is that the objects of $h(\mathbf{C},\mathcal{R})$ represent (reversible) histories. We shall now extend $h(\mathbf{C},\mathcal{R})$ with "reversed" computations with the effect that such histories can be undone.

Definition 4.3 [Category $h_\star(\mathbf{C},\mathcal{R})$ of reversible histories] Suppose that $\langle \mathcal{I},\mathcal{R}\rangle$ is a factorisation system. Let $\Phi\colon \mathbf{C} \to \mathbf{C}[\mathcal{R}^{-1}]$ be the canonical functor to the category of fractions. Let $h_\star(\mathbf{C},\mathcal{R})$ denote the category with:

- objects: arrows g in $\mathcal{R}$;
- arrows: formal diagrams, as illustrated, with $f \in \mathcal{I}$, $f_\star \in \mathbf{C}[\mathcal{R}^{-1}]$, such that $f_\star\Phi(g_1) = \Phi(g_2 f)$ in $\mathbf{C}[\mathcal{R}^{-1}]$.

$$\begin{array}{ccc} P_1 & \xrightarrow{f} & P_2 \\ {\scriptstyle g_1}\downarrow & & \downarrow{\scriptstyle g_2} \\ Q_1 & \xrightarrow[f_\star]{} & Q_2 \end{array}$$

There is an evident functor $\Psi\colon h(\mathbf{C},\mathcal{R}) \to h_\star(\mathbf{C},\mathcal{R})$ which maps the lower component of a history morphism from $\mathbf{C}$ to $\mathbf{C}[\mathcal{R}^{-1}]$ via Φ:

$$\begin{array}{ccc} P_1 & \xrightarrow{f'} & P_2 \\ {\scriptstyle g_1}\downarrow & & \downarrow{\scriptstyle g_2} \\ Q_1 & \xrightarrow[f]{} & Q_2 \end{array} \quad \longmapsto \quad \begin{array}{ccc} P_1 & \xrightarrow{f'} & P_2 \\ {\scriptstyle g_1}\downarrow & & \downarrow{\scriptstyle g_2} \\ Q_1 & \xrightarrow[\Phi f]{} & Q_2 \end{array}$$

Let $M_\star\colon h_\star(\mathbf{C},\mathcal{R}) \to \mathcal{I}$ be the functor which takes an arrow of $h_\star(\mathbf{C},\mathcal{R})$ to its upper component. Clearly $M_\star\Psi = M$.

Theorem 4.4 *$M_\star$ is an equivalence of categories.*

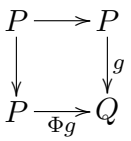
Proof. Let $N_\star = \Psi N\colon \mathcal{I} \to h_\star(\mathbf{C},\mathcal{R})$ (see Fig 2) – clearly $M_\star N_\star = \mathrm{id}_{\mathcal{I}}$, we shall show that there exists a natural isomorphism $N_\star M_\star \Rightarrow \mathrm{id}_{h_\star(\mathbf{C},\mathcal{R})}$.

Indeed, since Ψ is the identity on objects, we have $N_\star M_\star g = \Psi NM\Psi g = NMg$, and thus it suffices to show that $\Phi\epsilon$ is a natural isomorphism, where ϵ is the counit

of the adjunction $N \dashv M$. We illustrate $\Psi\epsilon_g$, clearly it is an invertible morphism of $h_\star(\mathbf{C}, \mathcal{R})$. Naturality is straightforward.

□

Recall from Example 2.5 that $\langle \mathbf{C}, \mathrm{Iso}(\mathbf{C})\rangle$ and $\langle \mathrm{Iso}(\mathbf{C}), \mathbf{C}\rangle$ are trivial factorisation systems in any category $\mathbf{C}$. The conclusion of Theorem 4.4 implies immediately that $h_*(\mathbf{C}, \mathrm{Iso}(\mathbf{C})) \simeq \mathbf{C}$ and $h_*(\mathbf{C}, \mathbf{C}) \simeq \mathrm{Iso}(\mathbf{C})$.

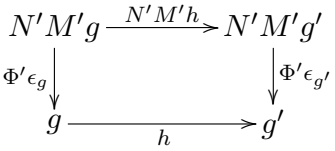


We shall now show that $h_\star(\mathbf{C}, \mathcal{R})$ essentially follows from a free construction. Let $\mathcal{R}' = \{\, \langle g, \varphi\rangle \in \mathrm{Ar}(h(\mathbf{C}, \mathcal{R})) \mid g \in \mathcal{R} \,\}$, the set of those arrows of $h(\mathbf{C}, \mathcal{R})$ where the lower component is in $\mathcal{R}$ (cf paragraph following Definition 4.1).

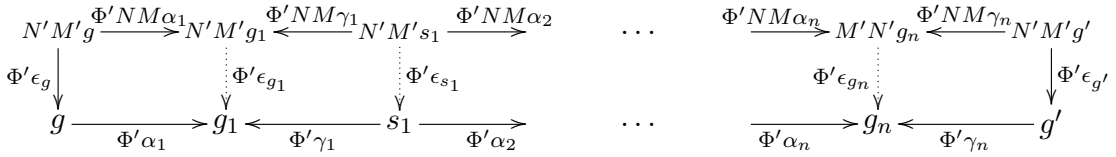
Theorem 4.5 *There is an equivalence of categories $h_\star(\mathbf{C}, \mathcal{R}) \simeq h(\mathbf{C}, \mathcal{R})[\mathcal{R}'^{-1}]$.*

Proof. Let $\mathbf{X} = h(\mathbf{C}, \mathcal{R})[\mathcal{R}'^{-1}]$. Since we know that $h_\star(\mathbf{C}, \mathcal{R}) \simeq \mathcal{I}$, it is enough to show that also $\mathbf{X} \simeq \mathcal{I}$. Let $\Phi' \colon h(\mathbf{C}, \mathcal{R}) \to \mathbf{X}$ be the canonical quotienting functor. Since $M \colon h(\mathbf{C}, \mathcal{R}) \to \mathcal{I}$ sends every member of $\mathcal{R}'$ to an isomorphism, we have a unique functor $M' \colon \mathbf{X} \to \mathcal{I}$ such that $M'\Phi' = M$. Let $N' = \Phi' N \colon \mathcal{I} \to \mathbf{X}$. Then $M'N' = M'\Phi' N = MN = \mathrm{id}_{\mathcal{I}}$.

Let $\epsilon \colon NM \to \mathrm{id}_{h(\mathbf{C},\mathcal{R})}$ be the counit of the adjunction $N \dashv M$. Clearly Φ' sends each component of ϵ to an isomorphism in $\mathbf{X}$. Since Φ' is the identity on objects, we have that for each object $g \in \mathbf{X}$, $\Phi'\epsilon_g \colon N'M'g \to g$ is an isomorphism. It remains to check that $\Phi'\epsilon$ defines a natural transformation $N'M' \to \mathrm{id}_{\mathbf{X}}$. To do this we need to check that the commutativity of an arbitrary square, as illustrated, where h is in $\mathbf{X}$.



It is well-known that arrows in $\mathbf{X}$ are equivalence classes of zig-zags in $h(\mathbf{C}, \mathcal{R})$ where each of the reverse arrows is in $\mathcal{R}'$. Using the functoriality of $N'M'$ and the fact that ϵ is a natural transformation, we can "fill in" the diagram below at each point, and since $h = (\Phi'\gamma_n)^{-1}\Phi'\alpha_n \ldots (\Phi'\gamma_1)^{-1}\Phi'\alpha_1$, naturality easily follows by a straightforward diagram chase.



□

Considering Examples 3.1, 3.2 and 3.5, Theorem 4.4 states that to understand the structure of causal computations it is enough to remember the maximal reversible component of a given computation and allow these histories to be backtracked.

Returning to the discussion concerning the net of Figure 1, the missing ingredient was clearly the explicit keeping track of the history of the current computation – ie instead of working in $\mathbf{C}_{\mathcal{N}}[\mathcal{R}^{-1}]$ we work in the history category $h_\star(\mathbf{C}_{\mathcal{N}}, \mathcal{R})$.

(cf Definition 4.3). Our main result, Theorem 4.4, establishes that the categories $h_\star(\mathbf{C}_\mathcal{N}, \mathcal{R})$ and $\mathcal{I}$ are equivalent, which confirms that the computations of nets with histories are essentially the same as the causal computations of the original net. Of course, the main result is clearly more general than this particular example, for instance it underlies the previous work on RCCS [1, 2].

Indeed, it is interesting to compare the concrete implementation of a reversible process algebra, like RCCS, with the abstract construction we present in this paper. Roughly, the definition of RCCS in [1] can be summed up as the development of a correct syntactic presentation of the category of reversible histories $h_\star(\mathbf{C}, \mathcal{R})$, where $\mathbf{C}$ is the category of computations of CCS.

5 Free categories as transition systems

The categories of Examples 3.2 and 3.5 can be thought of as a transition systems as well as categories; indeed, since the categories are generated freely, their arrows can be seen as (equivalence classes of) traces in the transition systems. Here we shall elucidate the consequences of our main Theorem 4.4 for the underlying transition systems, obtaining a direct generalisation of the main result of [2]. Notice however that the results of §4 are more general, since the underlying categories are not assumed to be free; indeed, the only assumption is the presence of a factorisation system.

Let $\mathcal{S} = \langle V, E\rangle$ be a tensor scheme with edges E partitioned into sets of irreversible actions I and reversible actions R. Let $\mathbf{C}$ be a freely generated SM category over $\mathcal{S}$. Let $\mathcal{R}$ be the subcategory of $\mathbf{C}$ generated by $\mathcal{S}_R = \langle V, R\rangle$. Let $\mathcal{I} = \mathcal{R}^\top$. Then $\langle \mathcal{I}, \mathcal{R}\rangle$ is a factorisation system. [3]

Definition 5.1 Let $\mathrm{TS}(\mathbf{C})$ be defined as follows:

- states are isomorphism classes of objects of $\mathbf{C}$;
- transitions are labelled with elements of E and arise as follows

$$\frac{P_1 \otimes P_2 \xrightarrow{\alpha \otimes P_2} P_1' \otimes P_2 \text{ in } \mathbf{C}, \quad \alpha \in E}{[P_1 \otimes P_2] \xrightarrow{\alpha} [P_1' \otimes P_2]}$$

Using the fact that $\mathbf{C}$ is freely generated, any non-invertible arrow of $\mathbf{C}$ generates a finite set of traces in $\mathrm{TS}(\mathbf{C})$. We shall refer to each possible trace of an arbitrary morphism f in $\mathbf{C}$ as a *serialisation* of f.

A trace σ is said to be *causal* if it is a serialisation of an arrow f in $\mathcal{I}$. A trace σ is an *i-transaction* if it is causal and contains precisely one action $i \in I$ (and arbitrarily many actions from R). Let $\mathrm{CTS}(\mathbf{C})$ be the LTS with the same states as $\mathrm{TS}(\mathbf{C})$, but with transitions

$$\frac{[P] \xrightarrow{\sigma} [P'] \text{ in } \mathrm{TS}(\mathbf{C}), \quad \sigma \text{ an } i\text{-transaction}}{[P] \xrightarrow{i} [P'] \text{ in } \mathrm{CTS}(\mathbf{C})}$$

[3] We leave it as future work to determine sufficient conditions on a subcategory which ensure that $\mathcal{R} = \mathcal{R}^{\top\perp}$.

Thus CTS($\mathbf{C}$) is the LTS of transactions. Correspondingly, we shall now define the history LTS, where states are enriched with a history, and the transitions are those of $TS(\mathbf{C})$ as well as new transitions which allow backtracking.

Definition 5.2 Let RTS($\mathbf{C}$) be defined as follows:

- states: isomorphism classes of objects $h(\mathbf{C}, \mathcal{R})$ (structural isomorphisms);
- transitions labelled with elements of $E \cup R_*$ where $R_\star = \{ r_\star \mid r \in R \}$. They are derived from morphisms in $h_\star(\mathbf{C}, \mathcal{R})$, as illustrated below:

$$\frac{\begin{array}{ccc} P_1 & \xrightarrow{f} & P_2 \\ {\scriptstyle g_1}\downarrow & & \downarrow{\scriptstyle g_2} \\ Q_1 \otimes Q_2 & \xrightarrow{\alpha \otimes Q_2} & Q_1' \otimes Q_2 \end{array} \quad , \quad \alpha \in E}{\left[\begin{array}{c} P_1 \\ {\scriptstyle g_1}\downarrow \\ Q_1 \otimes Q_2 \end{array}\right] \xrightarrow{\alpha} \left[\begin{array}{c} P_2 \\ \downarrow{\scriptstyle g_2} \\ Q_1' \otimes Q_2 \end{array}\right]} \qquad \frac{\begin{array}{ccc} P_1 & \xrightarrow{f} & P_2 \\ {\scriptstyle g_1}\downarrow & & \downarrow{\scriptstyle g_2} \\ Q_1 \otimes Q_2 & \xrightarrow{r^{-1} \otimes Q_2} & Q_1' \otimes Q_2 \end{array} \quad , \quad r \in R}{\left[\begin{array}{c} P_1 \\ {\scriptstyle g_1}\downarrow \\ Q_1 \otimes Q_2 \end{array}\right] \xrightarrow{r_\star} \left[\begin{array}{c} P_2 \\ \downarrow{\scriptstyle g_2} \\ Q_1' \otimes Q_2 \end{array}\right]}$$

It is clear from the construction of $h_\star(\mathbf{C}, \mathcal{R})$ that any morphism in $h_\star(\mathbf{C}, \mathcal{R})$ induces a set of serialisations (traces) in RTS($\mathbf{C}$).

Theorem 5.3 *Consider a free* SM *category* $\mathbf{C}$ *generated from a tensor scheme* $\mathcal{S} = \langle V, E \rangle$ *with* $E = I + R$*, together with an induced factorisation system* $\langle \mathcal{I}, \mathcal{R} \rangle$ *where* $\mathcal{R}$ *is the subcategory of* $\mathbf{C}$ *freely generated by* $\mathcal{S}_R = \langle V, R \rangle$*. Let* CTS($\mathbf{C}$) *be the* LTS *of transactions (cf Definition 5.1) and* RTS($\mathbf{C}$) *be the reversible* LTS *(cf Definition 5.2) where the reversible actions are considered to be silent. Then* CTS($\mathbf{C}$) $\approx$ RTS($\mathbf{C}$)*.*

Proof. We shall show that the (object part of the) functor $M_\star \colon h_\star(\mathbf{C}, \mathcal{R}) \to \mathcal{I}$ is actually a functional weak bisimulation.

Recall that $M_\star(P \xrightarrow{g} Q) = P$. Clearly $M_\star$ is well-defined as a function from states of RTS($\mathbf{C}$) to states of CTS($\mathbf{C}$). Suppose that there is a transition

$$[P \xrightarrow{g} Q] \xrightarrow{\alpha} [P' \xrightarrow{g'} Q'].$$

$$\begin{array}{ccc} P & \xrightarrow{f} & P' \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle g'} \\ Q & \xrightarrow[\alpha \otimes X]{} & Q' \end{array}$$

Then either $\alpha \in R$, in which case the transition is silent – we have $P' \cong P$ so we can counter with the empty trace.

$$\begin{array}{ccccc} P & \longrightarrow & P & \xrightarrow{f_i} & P' \\ {\scriptstyle g}\downarrow & & \downarrow & (\dagger) & \downarrow \\ Q & \xrightarrow[g_*]{} & P & \xrightarrow[f_i]{} & P' \end{array}$$

If $\alpha \notin R$ then we have the first diagram where f is in $\mathcal{I}$. Since we are in a free category, any serialisation of f must contain α as a unique action from I. Thus f leads to a trace in $TS(\mathbf{C})$ which is an α-transaction – ie we have a labelled transition $[P] \xrightarrow{\alpha} [P']$ in CTS($\mathbf{C}$).

Now consider an arbitrary transition $[P] \xrightarrow{i} [P']$. Let $P \xrightarrow{f_i} P'$ be the corresponding arrow in $\mathcal{I}$. Then in particular we have the square (†) in $h_*(\mathbf{C}, \mathcal{R})$, as illustrated in the second diagram. Let g_* be the inverse to g in $\mathbf{C}[\mathcal{R}^{-1}]$. Clearly i is the only irreversible action in any serialisation (in RTS($\mathbf{C}$)) of the combined second diagram, so we have a weak transition $[p \xrightarrow{g} q] \longrightarrow^* \xrightarrow{i} [p' \to p']$. □

6 Conclusion

The main contribution of this paper is the development of the underlying abstract concepts which become apparent when designing "reversed" versions of known formalisms, such as Petri nets or CCS. In particular, we show that the problem reduces to developing the particular syntactic representations (such as the concrete syntactic representation of histories in RCCS) of the reversible history category $h_*(\mathbf{C}, \mathcal{R})$. The fact that the resulting computations capture the intended causal behaviour can then be seen as a consequence of our Theorem 4.4, which is formalism independent. We hope that this conceptual clarification will be of use to designers of reversible formalisms.

Another contribution is the observation that breaking up a computation into irreversible-reversible components naturally leads to a factorisation system on the category of computations. As part of future work, we plan to study such factorisation systems in more detail. We also plan to explore connections with previous work on factorisation systems in rewriting theory [7].

References

[1] Danos, V. and J. Krivine, *Reversible communicating systems*, in: *Proceedings of Concur'04*, LNCS **3170** (2004), pp. 292–307.

[2] Danos, V. and J. Krivine, *Transactions in RCCS*, in: *Proceedings of Concur'05*, LNCS **3653** (2005), pp. 398–412.

[3] Danos, V., J. Krivine and F. Tarissan, *Self-assembling trees*, in: *SOS'06*, ENTCS (2006), to appear.

[4] Freyd, P. J. and G. M. Kelly, *Categories of continuous functors,* I, Journal of Pure and Applied Algebra **2** (1972), pp. 169–191.

[5] Gabriel, P. and M. Zisman, "Calculus of fractions and homotopy theory," Springer-Verlag, 1967.

[6] Joyal, A. and R. Street, *The geometry of tensor calculus,* I, Advances in Mathematics **88** (1991), pp. 55–112.

[7] Melliès, P.-A., *A factorisation theorem in rewriting theory*, in: *Proceedings of CTCS '97*, LNCS **1290** (1997), pp. 49–68.

[8] Meseguer, J. and U. Montanari, *Petri nets are monoids*, Information and computation **88** (1990), pp. 105–155.

[9] Milner, R., "A Calculus of Communicating Systems," LNCS **92**, Springer, 1980.

[10] Phillips, I. and I. Ulidowski, *Reversing algebraic process calculi*, in: *Proceedings of FoSSaCS '06*, LNCS **3921** (2006), pp. 246–260.

[11] Street, R., *Higher categories, strings, cubes and simplex equations*, Applied Categorical Structures **3** (1995), pp. 29–77.

Synchrony vs Asynchrony in Communication Primitives

Daniele Gorla[1]

Dip. di Informatica, Univ. di Roma "La Sapienza", Italy

Abstract

We study, from the expressiveness point of view, the impact of synchrony in the communication primitives that arise when combining together some common and useful programming features like arity of data, communication medium and possibility of pattern matching. For some primitives, we show how their synchronous version can be encoded in their asynchronous counterpart via a fully abstract encoding, thus proving that the two versions have the same expressive power. For the remaining primitives, we prove that no 'reasonable' encoding can exist, thus proving that synchrony adds expressiveness to the language.

Keywords: Expressiveness, Encodings, Communication Primitives, Process Calculi.

1 Introduction

One distinguishing feature of languages for concurrent systems is the choice of the communication primitives they use for inter-process exchange. These primitives can range from very skeletal ones [14,6] to more sophisticated and powerful programming constructs [11,16,2,7]. It is then natural to formally study and compare these primitives from the expressive power perspective. As a consequence, results in this research line illuminate the peculiarities of every primitive and, thus, they can be exploited to choose the 'right' primitive when designing new languages and formalisms.

In [12], we studied *asynchronous* communication primitives and the impact that some very common and useful programming features (like *arity of data*, *communication medium* and possibility of *pattern-matching*) have on their expressiveness. As a result, we came out with:

- eight languages (that, for the sake of uniformity, were small variants of the π-calculus [20]), whose communication primitives were obtained by combining the above mentioned features;
- and with a hierarchy of such languages, based on their relative expressive power.

[1] Email: gorla@di.uniroma1.it

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

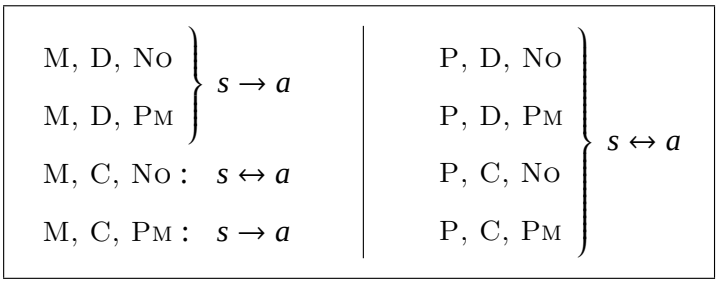


Fig. 1. The Impact of Synchrony in Communication Primitives

In this paper, we extend the results presented in [12] to assess, from the expressiveness point of view, the impact of *synchrony* in such primitives. Indeed, as we shall prove, the claim "for many purposes, synchronous message passing can be regarded as a special case of asynchronous message passing" [23] strongly relies on the accessory features that equip the communication primitives. In particular, for each of the eight languages studied in [12], we shall see whether their synchronous version has the same expressive power as their asynchronous counterpart or not. In the first case, we can freely implement the primitives asynchronously, since asynchrony usually poses fewer implementative problems; in the second case, asynchronous implementations are less innocuous.

Our results are summarised in Figure 1. There, M and P denote monadic/polyadic data exchanges; C and D denote channels/dataspaces; Pm and No denote presence/absence of pattern matching; s and a denote synchrony/asynchrony; finally, $s \rightarrow a$ means that the synchronous version of the primitive is strictly more expressive than its asynchronous counterpart, whereas $s \leftrightarrow a$ means that the two versions have the same expressive power.

To study the expressive power of a programming language, several techniques can be exploited. A first, very rough, test is to determine whether a language is Turing complete or not; however, since almost all 'useful' languages are Turing complete, this criterion is too coarse to compare different languages. A second, more informative, approach to show that a language is more expressive than another one is to find a problem that can be solved in the former under some conditions that cannot be met by any solution in the latter.

Another interesting approach to compare two languages consists in encoding one in the other (where an encoding is a function that translates terms of one language in terms of the other language) and studying the properties of the encoding functions. This is the approach we shall follow in this paper and it is very appealing for at least two reasons. First, it is a natural way to show how the key features of a language can be rendered in the other one. Second, it allows us to also carry out quantitative measures on language expressiveness: we can consider aspects like the size and the complexity of the encoding of a term w.r.t. the source term and, consequently, quantitatively assess the encoding proposed.

This paper is organised as follows. In Section 2, we start by comparing the impact of synchrony in the π-calculus [16]; in this way, we gently introduce the reader to the problem and sum up the main related achievements. In Section 3, we present the sixteen concurrent languages arising from the combination of the four features studied (synchrony, data arity, communication medium and presence of pattern-matching). In Section 4, we present some criteria that an encoding should satisfy to be a good means for language comparison. Then, in Section 5, we prove the results depicted in Figure 1; more precisely, we shall provide (*i*) a fully abstract encoding for all those languages whose synchronous and asynchronous versions have the same expressive power, and (*ii*) some intuitive reasons

on the impossibility for a 'reasonable' encoding for all those languages where synchrony improves expressiveness. Finally, Section 6 concludes the paper by discussing the results in Figure 1. Due to space limitation, proofs have been omitted from this extended abstract and can be found in an on-line longer version.

2 Synchrony and Asynchrony in the π-calculus

The π-calculus was originally equipped with synchronous, monadic and channel-based communication primitives [16]; a few years later, its asynchronous version appeared in literature [13,1] and became a reference point for its simplicity of distributed implementation [10,21]. Some effort has been spent to prove that the two formalisms have the same expressive power [13,1,22,4]; nowadays, it is widely believed that this is the case.

The idea underlying these encodings is that a synchronous exchange can be simulated by a sequence of asynchronous exchanges. As an example, consider the encodings from [13,1]:

	Honda and Tokoro's	Boudol's
$[\![\, \overline{a}\langle b\rangle.P \,]\!]$	$a(y).(\overline{y}\langle b\rangle \mid [\![\, P \,]\!])$	$(\nu c)(\overline{a}\langle c\rangle \mid c(y).(\overline{y}\langle b\rangle \mid [\![\, P \,]\!]))$
$[\![\, a(x).P \,]\!]$	$(\nu c)(\overline{a}\langle c\rangle \mid c(x).[\![\, P \,]\!])$	$a(z).(\nu d)(\overline{z}\langle d\rangle \mid d(x).[\![\, P \,]\!])$

where $\overline{a}\langle b\rangle.P$ denotes the output prefix (send b along a and, after reception, behave like P), $a(x).P$ denotes the input prefix (receive something from a and use it to replace x in the continuation P), $(\nu c)P$ denotes the restriction of c to P (c is accessible only from within P) and $P \mid Q$ denotes the parallel composition of processes P and Q.

These encodings are proved sound by exploiting some ad hoc techniques; e.g., Boudol only proves that his encoding is adequate w.r.t. a Morris-like preorder. On the other hand, [22,4] aim at stronger results for such an encoding: in particular, the first paper shows that it enjoys full abstraction w.r.t. a typed version of barbed equivalence [17], whereas the second paper proves full abstraction w.r.t. to may and fair testing [9,18] restricted to the translation of synchronous contexts. In both cases, it is necessary to reduce the observational power of the contexts since a context that does not abide by the protocol put forward by the encoding can easily break full abstraction. [2] In the first case, the type system characterises the respectful contexts, whereas in the second case the encoding itself yields them. Of course, the first alternative entails a stronger full abstraction result, because in general it accepts more contexts than the translated ones; however, it is usually much more complex. Thus, for the sake of simplicity, in this paper we shall adopt the second alternative; we strongly believe that all our full abstraction results could be also formulated in terms of typed equivalences, instead of translated equivalences.

Recently [5], it has been proved that there is no encoding of the synchronous π-calculus in its asynchronous version preserving *must testing* [9] and enjoying a few minimal properties. [3] This raises the problem of which equivalence should be adopted when defin-

[2] For example, processes $\overline{a}\langle b\rangle.\overline{a}\langle b\rangle$ and $\overline{a}\langle b\rangle \mid \overline{a}\langle b\rangle$ are equated both by barbed equivalence and by may/fair testing; nevertheless, $[\![\, \overline{a}\langle b\rangle.\overline{a}\langle b\rangle \,]\!]$ and $[\![\, \overline{a}\langle b\rangle \mid \overline{a}\langle b\rangle \,]\!]$ are not equivalent anymore. The problem is that $[\![\, \overline{a}\langle b\rangle \mid \overline{a}\langle b\rangle \,]\!]$ can exhibit two top-level outputs, whereas $[\![\, \overline{a}\langle b\rangle.\overline{a}\langle b\rangle \,]\!]$ only one; if the receiving context sends no acknowledgement back, the second output of $[\![\, \overline{a}\langle b\rangle.\overline{a}\langle b\rangle \,]\!]$ (that is blocked by the encoding of the first prefix) is never unleashed. The same problem holds for Honda and Tokoro's encoding, but with processes $a(x).a(y)$ and $a(x) \mid a(y)$.

[3] Another impossibility result is [19], but it relies on the interplay between output prefixes and non-deterministic choice.

ing the full abstraction property to assess expressiveness of two languages. As testified by the case of the π-calculus, such a choice is crucial, mainly when proving that a language $\mathcal{L}_1$ is more expressive than another language $\mathcal{L}_2$: every separation result based on a fixed equivalence could be criticised by saying that it actually compares not the expressive power of the languages, but the discriminating power of the equivalences. For this reason, to prove that $\mathcal{L}_1$ is at least as expressive as $\mathcal{L}_2$, we shall fix a set of minimal properties that every encoding should satisfy and prove that no encoding of $\mathcal{L}_2$ in $\mathcal{L}_1$ satisfying such properties exists.

3 A Family of Process Languages

Syntax. We assume a countable set of *names*, $\mathcal{N}$, ranged over by $a, b, x, y, n, m, \cdots$. Notationally, when a name is used as a channel, we shall prefer letters $a, b, c, \cdots$; when a name is used as an input variable, we shall prefer letters $x, y, z, \cdots$; to denote a generic name, we shall use letters $n, m, \cdots$. The (parametric) syntax of our languages is given in the upper part of Figure 2. The different languages are obtained by plugging into this basic syntax a proper definition for input prefixes (*IN*) and output processes (*OutProc*). As usual, **0** and $P|Q$ denote the terminated process and the parallel composition of two processes, resp.; $(\nu n)P$ restricts to P the visibility of n; finally, **if** $n = m$ **then** P and $!P$ are the standard constructs for name matching and process replication. [4]

In this paper, we study the synchronous/asynchronous versions of the primitives arising by the possible combinations of three features: *arity* (monadic vs. polyadic data), *communication medium* (channels vs. shared dataspaces) and *pattern-matching*. As a result, we have a family of sixteen languages, denoted as $L^{\beta_1}_{\beta_2,\beta_3,\beta_4}$, where

- $\beta_1 = \text{S}$, if we have synchronous communications, and $\beta_1 = \text{A}$, otherwise;
- $\beta_2 = \text{P}$, if we have polyadic data, and $\beta_2 = \text{M}$, otherwise;
- $\beta_3 = \text{C}$, if we have channel-based communications, and $\beta_3 = \text{D}$, otherwise;
- $\beta_4 = \text{PM}$, if we have pattern-matching, and $\beta_4 = \text{NO}$, otherwise.

Now, the full syntax of every language is obtained from the productions in the lower part of Figure 2. There, $\widetilde{\ \cdot\ }$ denotes a (possibly empty) sequence of elements of kind $\cdot$; whenever useful, we shall write a tuple $\widetilde{\ \cdot\ }$ as the sequence of its elements, separated by a comma; sometimes, we shall also consider tuples simply as sets. Templates of kind x are called *formal* and can be replaced by every name upon withdrawal of a datum; templates of kind $\ulcorner n \urcorner$ are called *actual* and impose that the datum withdrawn contains exactly name n. As usual, $a(\cdots, x, \cdots).P$ and $(\nu x)P$ bind x in P; the corresponding notions of free and bound names of a process, $\text{FN}(P)$ and $\text{BN}(P)$, and of alpha-conversion, $=_\alpha$, are assumed. We let $\text{N}(P)$ denote $\text{FN}(P) \cup \text{BN}(P)$.

Notice that in $L^{\widetilde{\ \cdot\ }}_{\widetilde{\ \cdot\ },\text{PM}}$ the **if-then** construct is redundant because it can be implemented via pattern matching; we kept it for the sake of uniformity with the other languages. Finally, notice that $L^{\text{A}}_{\widetilde{\ \cdot\ },\widetilde{\ \cdot\ }}$ can be seen as the sub-language of $L^{\text{S}}_{\widetilde{\ \cdot\ },\widetilde{\ \cdot\ }}$ where every output prefix is followed by a **0** continuation. Thus, the non-trivial contribution of this work is in giving a converse encoding, or in proving that this cannot exist.

Operational semantics. The operational semantics of the languages is given by means of

[4] Notice that, for the sake of simplicity, we used here replication and a **if-then** construct instead of recursion and the more powerful **if-then-else** used in [12]; this choice does not undermine all our results that still hold also with the other operators.

Basic Processes:

$$P,Q,R \quad ::= \quad \mathbf{0} \;\Big|\; \mathit{OutProc} \;\Big|\; \mathit{IN}.P \;\Big|\; (\nu n)P \;\Big|\; P|Q \;\Big|\; \textbf{if } n = m \textbf{ then } P \;\Big|\; !P$$

$L^{\mathrm{A}}_{\text{-},\text{-},\text{-}}$: $\mathit{OutProc} ::= \mathit{OUT}$

$L^{\mathrm{S}}_{\text{-},\text{-},\text{-}}$: $\mathit{OutProc} ::= \mathit{OUT}.P$

$L^{\text{-}}_{\mathrm{M,D,NO}}$:	$P,Q,R ::= \ldots$	$\mathit{IN} ::= (x)$	$\mathit{OUT} ::= \langle b \rangle$
$L^{\text{-}}_{\mathrm{M,D,PM}}$:	$P,Q,R ::= \ldots$	$\mathit{IN} ::= (T)$	$\mathit{OUT} ::= \langle b \rangle$
$L^{\text{-}}_{\mathrm{M,C,NO}}$:	$P,Q,R ::= \ldots$	$\mathit{IN} ::= a(x)$	$\mathit{OUT} ::= \overline{a}\langle b \rangle$
$L^{\text{-}}_{\mathrm{M,C,PM}}$:	$P,Q,R ::= \ldots$	$\mathit{IN} ::= a(T)$	$\mathit{OUT} ::= \overline{a}\langle b \rangle$
$L^{\text{-}}_{\mathrm{P,D,NO}}$:	$P,Q,R ::= \ldots$	$\mathit{IN} ::= (\widetilde{x})$	$\mathit{OUT} ::= \langle \widetilde{b} \rangle$
$L^{\text{-}}_{\mathrm{P,D,PM}}$:	$P,Q,R ::= \ldots$	$\mathit{IN} ::= (\widetilde{T})$	$\mathit{OUT} ::= \langle \widetilde{b} \rangle$
$L^{\text{-}}_{\mathrm{P,C,NO}}$:	$P,Q,R ::= \ldots$	$\mathit{IN} ::= a(\widetilde{x})$	$\mathit{OUT} ::= \overline{a}\langle \widetilde{b} \rangle$
$L^{\text{-}}_{\mathrm{P,C,PM}}$:	$P,Q,R ::= \ldots$	$\mathit{IN} ::= a(\widetilde{T})$	$\mathit{OUT} ::= \overline{a}\langle \widetilde{b} \rangle$

where $T ::= x \;\Big|\; \ulcorner n \urcorner$ (*Template*)

Fig. 2. Syntax of the 16 Languages

a *labelled transition system* (LTS) describing the actions a process can perform to evolve. Judgements take the form $P \xrightarrow{\alpha} P'$, meaning that P can become P' upon exhibition of label α. *Labels* take the form

$$\alpha \quad ::= \quad \tau \;\Big|\; a?\widetilde{b} \;\Big|\; (\nu \widetilde{c})a!\widetilde{b} \;\Big|\; ?\widetilde{b} \;\Big|\; (\nu \widetilde{c})!\widetilde{b}$$

Traditionally, τ denotes an internal computation; $a?\widetilde{b}$ and $(\nu \widetilde{c})a!\widetilde{b}$ denote the reception/sending of a sequence of names $\widetilde{b}$ along channel a; when channels are not present, $?\widetilde{b}$ and $(\nu \widetilde{c})!\widetilde{b}$ denote the withdrawal/emission of $\widetilde{b}$ from/in the shared dataspace. In $(\nu \widetilde{c})a!\widetilde{b}$ and $(\nu \widetilde{c})!\widetilde{b}$, some of the sent names, viz. $\widetilde{c}$ ($\subseteq \widetilde{b}$), are restricted. Notationally, $(\nu \widetilde{c})^{\text{-}}!\widetilde{b}$ stands for either $(\nu \widetilde{c})a!\widetilde{b}$ or $(\nu \widetilde{c})!\widetilde{b}$; similarly, $^{\text{-}}?\widetilde{b}$ stands for either $a?\widetilde{b}$ or $?\widetilde{b}$. As usual, $\mathrm{BN}((\nu \widetilde{c})^{\text{-}}!\widetilde{b}) \triangleq \widetilde{c}$; $\mathrm{FN}(\alpha)$ and $\mathrm{N}(\alpha)$ are defined accordingly.

The LTS provides some rules shared by all the languages; the different semantics are obtained from the axioms for input/output actions. The LTS relies on π-calculus structural equivalence, $\equiv$, that rearranges a process to let it evolve according to the rules of the LTS and that is defined by the following standard axioms [20]:

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$

$$!P \equiv P \mid !P \qquad \textbf{if } n = n \textbf{ then } P \equiv P \qquad P \equiv P' \text{ if } P =_{\alpha} P'$$

$$(\nu n)\mathbf{0} \equiv \mathbf{0} \qquad (\nu n)(\nu m)P \equiv (\nu m)(\nu n)P \qquad P \mid (\nu n)Q \equiv (\nu n)(P \mid Q) \text{ if } n \notin \mathrm{FN}(P)$$

The common rules of the LTS are reported below (since they are an easy adaptation of an early-style LTS for the π-calculus, we do not comment on them and refer the interested reader to [20]):

$$\frac{P \xrightarrow{?\widetilde{b}} P' \qquad Q \xrightarrow{!\widetilde{b}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \frac{P \xrightarrow{a?\widetilde{b}} P' \qquad Q \xrightarrow{a!\widetilde{b}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad n \notin \mathrm{N}(\alpha)}{(\nu n)P \xrightarrow{\alpha} (\nu n)P'} \qquad \frac{P \xrightarrow{(\nu \widetilde{c})\text{-}!\widetilde{b}} P' \qquad n \in \widetilde{b} \setminus \{\text{-}, \widetilde{c}\}}{(\nu n)P \xrightarrow{(\nu n, \widetilde{c})\text{-}!\widetilde{b}} P'}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad \mathrm{BN}(\alpha) \cap \mathrm{FN}(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \frac{P \equiv P_1 \xrightarrow{\alpha} P_2 \equiv P'}{P \xrightarrow{\alpha} P'}$$

To define the semantics for the basic actions of the various languages, we must specify when a template matches a datum. Intuitively, this happens whenever both have the same length and corresponding fields match (i.e., $\ulcorner n \urcorner$ matches n and x matches every name). This can be formalised via a partial function, called *pattern-matching* and written MATCH, that also returns a substitution σ; the latter will be applied to the process that performed the input to replace formal templates with the corresponding names of the datum retrieved. These intuitions are formalised by the following rules:

$$\mathrm{MATCH}(\,;\,) = \epsilon \qquad \mathrm{MATCH}(\ulcorner n \urcorner; n) = \epsilon \qquad \mathrm{MATCH}(x; n) = \{^{n}/_{x}\}$$

$$\frac{\mathrm{MATCH}(T; b) = \sigma_1 \qquad \mathrm{MATCH}(\widetilde{T}; \widetilde{b}) = \sigma_2}{\mathrm{MATCH}(T, \widetilde{T}\,;\, b, \widetilde{b}) = \sigma_1 \circ \sigma_2}$$

where 'ϵ' denotes the empty substitution and '$\circ$' denotes substitution composition. Now, the operational rules for input actions in languages $L^{\text{-}}_{\text{-,D,-}}$ and $L^{\text{-}}_{\text{-,C,-}}$ are

$$(\widetilde{T}).P \xrightarrow{?\widetilde{b}} P\sigma \qquad a(\widetilde{T}).P \xrightarrow{a?\widetilde{b}} P\sigma$$

whenever $\mathrm{MATCH}(\widetilde{T}\,;\,\widetilde{b}) = \sigma$. Symmetrically, the rules for output actions in languages $L^{\mathrm{A}}_{\text{-,D,-}}$, $L^{\mathrm{A}}_{\text{-,C,-}}$, $L^{\mathrm{S}}_{\text{-,D,-}}$ and $L^{\mathrm{S}}_{\text{-,C,-}}$ are, respectively,

$$\langle\widetilde{b}\rangle \xrightarrow{!\widetilde{b}} \mathbf{0} \qquad \overline{a}\langle\widetilde{b}\rangle \xrightarrow{a!\widetilde{b}} \mathbf{0} \qquad \langle\widetilde{b}\rangle.P \xrightarrow{!\widetilde{b}} P \qquad \overline{a}\langle\widetilde{b}\rangle.P \xrightarrow{a!\widetilde{b}} P$$

Notation: A substitution σ is a finite partial mapping of names for names; $P\sigma$ denotes the (capture avoiding) application of σ to P. As usual, we let $\Rightarrow$ stand for the reflexive and transitive closure of $\xrightarrow{\tau}$, $\stackrel{\alpha}{\Longrightarrow}$ stand for $\Rightarrow\xrightarrow{\alpha}\Rightarrow$ and $\xrightarrow{\tau}{}^{k}$ denote a sequence of k τ-steps. We shall write $P \xrightarrow{\alpha}$ to mean that there exists a process P' such that $P \xrightarrow{\alpha} P'$; a similar notation is adopted for $P \Rightarrow$ and $P \stackrel{\alpha}{\Longrightarrow}$. Moreover, we let ϕ range over visible actions (i.e. labels different from τ) and ρ to range over (possibly empty) sequences of visible actions. Formally, $\rho ::= \varepsilon \mid \phi \cdot \rho$, where '$\varepsilon$' denotes the empty sequence of

actions and '$\cdot$' represents concatenation; then, $N \stackrel{\varepsilon}{\Longrightarrow}$ is defined as $N \Rightarrow$ and $N \stackrel{\phi\cdot\rho}{\Longrightarrow}$ is defined as $N \stackrel{\phi}{\Longrightarrow}\stackrel{\rho}{\Longrightarrow}$.

4 Quality of an Encoding

We now compare the synchronous and the asynchronous version of the communication primitives just presented by trying to encode every synchronous language in its asynchronous version. Formally, an *encoding* $[\![\cdot]\!]$ is a function mapping terms of the source language into terms of the target language. As already said, the relative expressive power of our languages can be established by defining some criteria to evaluate the quality of the encodings or to prove impossibility results.

Roughly speaking, the encoding must not change the semantics of a source term, i.e. it must preserve the observable behaviour of the term without introducing new behaviours. This means that the encoded term and the source one should be engageable in the same kinds of interactions and that aspects like deadlock and divergence are either present in both terms or in neither of them. We now discuss two possible ways of formalising this requirement. The first one, called *full abstraction*, is usually exploited for encodability results; the second one, called *reasonableness*, is usually exploited in the impossibility results.

Full abstraction. When a language can be encoded in another one, we shall prove that the encoding function enjoys *full abstraction* w.r.t. barbed equivalence restricted to translated contexts. This is a satisfying result since (weak) barbed equivalence is often considered to be the 'touchstone' semantic theory for several process languages. *Barbed equivalence* is obtained by closing under name restriction and parallel composition a relation called *barbed bisimilarity*, that equates two terms that offer the same observable behaviour along all possible computations.

In our framework, a *context* $C[\cdot]$ is a process built up from a hole $[\cdot]$ (to be filled with any process) by using parallel composition and restriction. Formally,

$$C[\cdot] \quad ::= \quad [\cdot] \quad \Big| \quad P \mid C[\cdot] \quad \Big| \quad (\nu n)C[\cdot]$$

Definition 4.1 [Barbs] [5]

- $P \downarrow_{OUT^k}$ holds true iff $P \xrightarrow{(\nu\tilde{c})!\tilde{b}}$ and $|\tilde{b}| = k$; $P \downarrow_{OUT_a}$ holds true iff $P \xrightarrow{(\nu\tilde{c})a!\tilde{b}}$.
- $P \downarrow_{IN^k}$ holds true iff $P \xrightarrow{?\tilde{b}}$ and $|\tilde{b}| = k$; $P \downarrow_{IN_a}$ holds true iff $P \xrightarrow{a?\tilde{b}}$.
- Let o range over $\{OUT^k, OUT_a, IN^k, IN_a\}$; then, $P \Downarrow_o$ stands for $\exists P'.P \Rightarrow P' \downarrow_o$.

Definition 4.2 [Barbed Bisimilarity and Equivalence] A symmetric relation $\mathfrak{R}$ between processes is a *barbed bisimulation* if, for every $(P, Q) \in \mathfrak{R}$, it holds that

(i) $P \downarrow_o$ implies $Q \Downarrow_o$, and
(ii) $P \xrightarrow{\tau} P'$ implies $Q \Rightarrow Q'$, for some Q' such that $(P', Q') \in \mathfrak{R}$.

[5] In order to obtain meaningful equivalences, barbs in $L_{\tilde{\text{M,D,-}}}$ should be defined by also specifying the argument of the action. However, since we shall not give full abstraction results for such languages, we ignore this aspect. By the way, notice that for languages $L_{\tilde{\text{P,D,-}}}$ the arguments of the action are not strictly necessary, since the barbed equivalence arising from this different kind of barbs would coincide with $\cong$ defined here.

Barbed bisimilarity, $\dot{\cong}$, is the largest barbed bisimulation. P and Q are *barbed equivalent*, written $P \cong Q$, if and only if $C[P] \dot{\cong} C[Q]$, for every context $C[\cdot]$.

As already said in Section 2, a good form of full abstraction for a given encoding $[\![\cdot]\!] : \mathcal{L}_1 \to \mathcal{L}_2$ is w.r.t. *translated observers*, i.e. observers that abide by the schema imposed by the encoding function. Thus, we now restrict the equivalences introduced so far to keep this choice into account: first, not all the barbs from Definition 5 can be observed by a translated observer; second, we only need to consider translated contexts when defining barbed equivalence. The following definition formalises these ideas; there, we say that an action α performed by a $\mathcal{L}_2$-process can be consumed by the translation of a $\mathcal{L}_1$-process R if $[\![R]\!] \stackrel{\rho}{\Longrightarrow} \stackrel{\alpha'}{\longrightarrow}$, with α' *synchronisable* with α (i.e., with $\alpha' = \overline{?\widetilde{b}}$ and $\alpha = (\nu\widetilde{c})\overline{!\widetilde{b}}$, or vice versa) and $\mathrm{BN}(\rho) \cap \mathrm{N}(\alpha) = \emptyset$.

Definition 4.3 [Translated Barbed Bisimilarity and Equivalence] Fix an encoding $[\![\cdot]\!] : \mathcal{L}_1 \to \mathcal{L}_2$.

- Let P be a $\mathcal{L}_2$-process; $P \downarrow_o^{tr}$ holds true iff $P \downarrow_o$ with an action that can be consumed by the translation of some $\mathcal{L}_1$-process; $P \Downarrow_o^{tr}$ is defined accordingly.
- A symmetric relation $\mathfrak{R}$ between $\mathcal{L}_2$-processes is a *translated barbed bisimulation* if, for every $(P, Q) \in \mathfrak{R}$, it holds that
  (i) $P \downarrow_o^{tr}$ implies $Q \Downarrow_o^{tr}$, and
  (ii) $P \stackrel{\tau}{\longrightarrow} P'$ implies $Q \Rightarrow Q'$, for some Q' such that $(P', Q') \in \mathfrak{R}$.
  Translated barbed bisimilarity, $\dot{\cong}^{tr}$, is the largest translated barbed bisimulation.
- P and Q are *translated barbed equivalent*, written $P \cong^{tr} Q$, if and only if $C[P] \dot{\cong}^{tr} C[Q]$, for every context $C[\cdot]$ resulting from the translation of a $\mathcal{L}_1$-context via $[\![\cdot]\!]$ extended with $[\![[\cdot]]\!] \triangleq [\cdot]$.

Reasonable Encoding. To prove that two languages have different expressive power, we shall leave full abstraction out (since it requires to fix an equivalence relation): instead, we shall collect together some 'reasonable' requirements and prove that no encoding function satisfying them exists. The main requirement is *faithfulness*: the encoding must preserve and reflect the barbs (i.e., the encoding should maintain all the original barbs without introducing new ones); moreover, it should also preserve and reflect divergence. However, these two requirements alone are not enough to control deadlock. Thus, we shall also require that the computations of a process correspond to the computations of its encoding, and vice versa; this property is usually known as *operational correspondence*. Furthermore, a good encoding cannot depend on the particular names involved in the source process, since we are dealing with a family of name-passing languages; we call this property *name invariance*. Finally, the encoding should not decrease the degree of parallelism in favour of centralised entities that control the behaviour of the encoded term; we express this last property as *homomorphism w.r.t.* '|'.

Definition 4.4 [Reasonable Encoding] An encoding $[\![\cdot]\!]$ is *reasonable* if it enjoys the following properties:

(i) *(homomorphism w.r.t. '|'):* $[\![P_1 | P_2]\!] \triangleq [\![P_1]\!] \mid [\![P_2]\!]$.

(ii) *(name invariance):* $[\![P\sigma]\!] \triangleq [\![P]\!]\sigma$, for every permutation of source language names σ.

(iii) *(faithfulness):* $P \Downarrow_o$ iff $[\![P]\!] \Downarrow_{o'}$; P diverges iff $[\![P]\!]$ diverges.

(iv) *(operational correspondence):*
  (a) if $P \Rightarrow P'$ then $[\![P]\!] \Rightarrow [\![P']\!]$;
  (b) if $[\![P]\!] \Rightarrow Q$ then there exists a P' such that $P \Rightarrow P'$ and $Q \Rightarrow [\![P']\!]$.

Evaluation criteria. To sum up, for our encodability results we aim at proving that the encoding function does not introduce divergence and that it enjoys full abstraction w.r.t. translated barbed equivalence; on the other hand, we shall establish our impossibility results by proving that no reasonable encoding exists. Usually, the latter proofs are by contradiction: we assume that a reasonable encoding exists and show that it cannot be reasonable. This can require a lot of work. However, in this paper, we shall exploit the simple proof-technique developed in [12]: exhibit a process that cannot reduce but whose encoding reduces. This fact, together with operational correspondence, implies that the encoding introduces divergence.

Proposition 4.5 *Let P be a process such that $P \stackrel{\tau}{\not\longrightarrow}$ but $[\![P]\!] \stackrel{\tau}{\longrightarrow}$; then, $[\![\cdot]\!]$ is not reasonable.*

5 The Impact of Synchrony in Communication Primitives

In this section, we first consider those languages in which synchrony does not play a crucial rôle, i.e. those primitives whose synchronous and asynchronous versions have the same expressive power. We then analyse those primitives in which the presence of synchrony matters, i.e. those primitives whose asynchronous version is less expressive than the synchronous one.

$L^{\mathrm{S}}_{\mathrm{M,C,NO}}$ and $L^{\mathrm{A}}_{\mathrm{M,C,NO}}$ have the same expressive power. Easily, Boudol's encoding [1] can be used to prove that $L^{\mathrm{S}}_{\mathrm{M,C,NO}}$ is encodable in $L^{\mathrm{A}}_{\mathrm{M,C,NO}}$ with an encoding function that does not introduce divergence (trivially) and that enjoys full abstraction w.r.t. translated barbed equivalence (see [22]).

$L^{\mathrm{S}}_{\mathrm{P,C,PM}}$ and $L^{\mathrm{A}}_{\mathrm{P,C,PM}}$ have the same expressive power. To prove that $L^{\mathrm{S}}_{\mathrm{P,C,PM}}$ can be reasonably encoded in $L^{\mathrm{A}}_{\mathrm{P,C,PM}}$, it suffices to impose that the first name of every datum is a restricted channel used to unleash the continuation of the output prefix; conversely, every template starts with a new variable over which an acknowledgement is sent upon reception of the datum. This discipline is rendered by the following encoding:

$$[\![\overline{a}\langle\widetilde{b}\rangle.P]\!] \triangleq (\nu c)(\overline{a}\langle c, \widetilde{b}\rangle \mid c().[\![P]\!])$$

$$[\![a(\widetilde{T}).P]\!] \triangleq a(x, \widetilde{T}).(\overline{x}\langle\rangle \mid [\![P]\!])$$

for c and x fresh.

The encoding just presented is satisfying because it does not introduce divergence and enjoys full abstraction, as proved in the following theorem.

Theorem 5.1 *The encoding $[\![\cdot]\!] : L^{\mathrm{S}}_{\mathrm{P,C,PM}} \longrightarrow L^{\mathrm{A}}_{\mathrm{P,C,PM}}$ does not introduce divergence; moreover, $P \cong Q$ if and only if $[\![P]\!] \cong^{tr} [\![Q]\!]$.*

$L^{\mathrm{S}}_{\mathrm{P,C,NO}}$ and $L^{\mathrm{A}}_{\mathrm{P,C,NO}}$ have the same expressive power. This result is an easy corollary of the encodability of $L^{\mathrm{S}}_{\mathrm{P,C,PM}}$ in $L^{\mathrm{A}}_{\mathrm{P,C,PM}}$: it suffices to restrict both the domain and the range of the encoding function to the sub-calculi of $L^{\mathrm{S}}_{\mathrm{P,C,PM}}$ and $L^{\mathrm{A}}_{\mathrm{P,C,PM}}$ with templates made up only by formal fields.

$L^{\mathrm{S}}_{\mathrm{P,D,PM}}$ and $L^{\mathrm{A}}_{\mathrm{P,D,PM}}$ have the same expressive power. To prove that $L^{\mathrm{S}}_{\mathrm{P,D,PM}}$ can be encoded in $L^{\mathrm{A}}_{\mathrm{P,D,PM}}$, consider the following translation:

$$[\![\, \langle b_1, \ldots, b_k \rangle.P \,]\!] \triangleq (\nu c)(\langle c, c, b_1, \ldots, b_k \rangle \mid (\ulcorner c \urcorner).[\![\, P \,]\!])$$

$$[\![\, (T_1, \ldots, T_k).P \,]\!] \triangleq (x, y, T_1, \ldots, T_k).(\langle x \rangle \mid [\![\, P \,]\!])$$

where c, x and y are fresh names. Intuitively, data of length one in a translated term are 'auxiliary' messages used as acknowledgements (ack, for short), to activate the continuation of an output action. The translation of output prefixes guarantees that 'actual' data in the source term are translated to data whose length is at least two; this clear distinction ensures us that no interference between an 'actual' data exchange and an 'auxiliary' ack exchange can ever happen. Moreover, the fact that acks rely on restricted names rules out interferences between different acks.

Theorem 5.2 *The encoding $[\![\, \cdot \,]\!] : L^{\mathrm{S}}_{\mathrm{P,D,PM}} \longrightarrow L^{\mathrm{A}}_{\mathrm{P,D,PM}}$ does not introduce divergence; moreover, $P \cong Q$ if and only if $[\![\, P \,]\!] \cong^{tr} [\![\, Q \,]\!]$.*

$L^{\mathrm{S}}_{\mathrm{P,D,NO}}$ and $L^{\mathrm{A}}_{\mathrm{P,D,NO}}$ have the same expressive power. Let us define the following notation: $\langle \overset{k}{\ldots} \rangle$ denotes $\langle b_1, \ldots, b_k \rangle$, where the b_i's are any names; similarly, $(\overset{k}{\ldots})$ denotes $(x_1, \ldots, x_k)$, where the x_i's are pairwise and distinct names. Now, consider the following encoding of $L^{\mathrm{S}}_{\mathrm{P,D,NO}}$ in $L^{\mathrm{A}}_{\mathrm{P,D,NO}}$:

$$[\![\, \langle b_1, \ldots, b_k \rangle.P \,]\!] \triangleq \langle \overset{4k+1}{\ldots} \rangle \mid (\overset{4k+2}{\ldots}).(\langle b_1, b_1, b_1, b_1, \cdots, b_k, b_k, b_k, b_k \rangle \mid (\overset{4k+3}{\ldots}).[\![\, P \,]\!])$$

$$[\![\, (x_1, \ldots, x_k).P \,]\!] \triangleq (\overset{4k+1}{\ldots}).(\langle \overset{4k+2}{\ldots} \rangle \mid (x_1, y_1, w_1, z_1, \cdots, x_k, y_k, w_k, z_k).(\langle \overset{4k+3}{\ldots} \rangle \mid [\![\, P \,]\!]))$$

for $y_1, w_1, z_1, \ldots, y_k, w_k, z_k$ fresh and pairwise distinct names and with the input variables in $(\overset{4k+1}{\ldots})$, $(\overset{4k+2}{\ldots})$ and $(\overset{4k+3}{\ldots})$ fresh for the continuation process. Intuitively, data of arity $4k$ within translated terms correspond to actual source data; data of arity $4k + 1$, $4k + 2$ and $4k + 3$ are, instead, only used for synchronisation purposes. In particular, an exchange of arity $4k + 1$ (that, from now on will be called *preliminary*) intuitively means "a datum of arity k is available"; an exchange of arity $4k+2$ (that, from now on will be called *initial*) intuitively means "a datum of arity k is going to be consumed"; finally, an exchange of arity $4k + 3$ (that, from now on will be called *final*) intuitively means "a datum of arity k has been consumed". Consumption of a k-ary source level datum happens within a $4k$-ary exchange (that, from now on will be called *consumptive*).

Of course, it is easy to have interferences between the auxiliary data introduced by the encoding of different processes, but this does not create any problem since such data only depend on the length of the translated actions. Consider, e.g., the encoding of the $L^{\mathrm{S}}_{\mathrm{P,D,NO}}$-process $(x).P \mid \langle b \rangle \mid (y).Q \mid \langle c \rangle$ and the reduction that replaces x with b in P and y with c in

Q. It is immaterial which of the two 5-ary 'preliminary' data (either the one from $[\![\langle b \rangle]\!]$ or the one from $[\![\langle c \rangle]\!]$) is accessed by $[\![(x).P]\!]$, since these are top-level asynchronous outputs and the names appearing in it are irrelevant. A similar argument holds also for the 'initial' 6-ary and the 'final' 7-ary data.

We believe that also this encoding enjoys full abstraction w.r.t. translated barbed equivalence; however, because of the interferences just discussed, we have still not been able to prove this result, though no counter-example against this conjecture has emerged yet. We leave this aspect for future work; for the moment, we prove the (not trivial) reasonableness of this encoding and argue that $L^{\mathrm{S}}_{\mathrm{P,D,NO}}$ and $L^{\mathrm{A}}_{\mathrm{P,D,NO}}$ have a comparable expressive power.

Lemma 5.3 *Let* $[\![P]\!] \xrightarrow{\tau}{}^{n_p+n_i+n_c+n_f} Q$, *where* $n_p/n_i/n_c/n_f$ *are the number of preliminary/initial/consumptive/final steps in the reduction from* $[\![P]\!]$ *into* Q. *Then,*

$$\begin{aligned} Q \equiv (\nu\widetilde{n})(& \textstyle\prod_{h=1}^{n_p-n_i}((\overset{4k_h+2}{\cdots}).(\langle \overset{4k_h}{\cdots} \rangle \mid (\overset{4k_h+3}{\cdots}).[\![P_h^1]\!]) \mid \langle \overset{4k_h+2}{\cdots} \rangle \mid (\overset{4k_h}{\cdots}).(\langle \overset{4k_h+3}{\cdots} \rangle \mid [\![Q_h^1]\!])) \mid \\ & \textstyle\prod_{j=1}^{n_i-n_c}(\langle \overset{4k_j}{\cdots} \rangle \mid (\overset{4k_j+3}{\cdots}).[\![P_j^2]\!] \mid (\overset{4k_j}{\cdots}).(\langle \overset{4k_j+3}{\cdots} \rangle \mid [\![Q_j^2]\!])) \mid \\ & \textstyle\prod_{m=1}^{n_c-n_f}((\overset{4k_m+3}{\cdots}).[\![P_m^3]\!] \mid \langle \overset{4k_m+3}{\cdots} \rangle) \mid [\![R]\!]) \end{aligned}$$

Lemma 5.4 *If* $[\![P]\!] \xrightarrow{\tau}{}^{n} Q$, *then* $P \xrightarrow{\tau}{}^{n_p} P'$ *and* $Q \xrightarrow{\tau}{}^{4n_p-n} [\![P']\!]$, *where* n_p *is the number of preliminary steps in the reduction from* $[\![P]\!]$ *into* Q.

Proposition 5.5 *The encoding* $[\![\cdot]\!] : L^{\mathrm{S}}_{\mathrm{P,D,NO}} \longrightarrow L^{\mathrm{A}}_{\mathrm{P,D,NO}}$ *is reasonable.*

Proof. By Lemma 5.4, both divergence freedom and operational correspondence are easy to prove; the remaining requirements are trivial. □

$L^{\mathrm{S}}_{\mathrm{M,D,NO}}$ is more expressive than $L^{\mathrm{A}}_{\mathrm{M,D,NO}}$.

Theorem 5.6 *There exists no reasonable encoding of* $L^{\mathrm{S}}_{\mathrm{M,D,NO}}$ *in* $L^{\mathrm{A}}_{\mathrm{M,D,NO}}$.

Proof. Consider the process $P \triangleq \langle a \rangle.\langle b \rangle$. If every trace of $[\![P]\!]$ was made up only by output labels, then it would be easy to prove that $\langle a \rangle.\langle b \rangle$ and $\langle b \rangle.\langle a \rangle$ have the same traces; thus, $[\![P \mid (x).\mathbf{if}\ x = b\ \mathbf{then}\ \Omega]\!]$ would diverge, since $[\![\langle b \rangle.\langle a \rangle \mid (x).\mathbf{if}\ x = b\ \mathbf{then}\ \Omega]\!]$ must diverge (here and in what follows, Ω denotes a divergent process). So, there exists a trace of $[\![P]\!]$ with at least an input label in the trace; let $\rho_1 \cdot ?n \cdot \rho_2$ be any of such traces and let $?n$ be the first input label. Notice that, by barb preservation, ρ_1 cannot be empty and must contain at least an output label $(\nu\widetilde{m})!m$; by definition of the LTS and asynchrony, $[\![P]\!] \xrightarrow{?m}$ that, again by asynchrony, implies that $[\![P]\!] \xrightarrow{\tau}$. By Proposition 4.5, $[\![\cdot]\!]$ cannot be reasonable. □

$L^{\mathrm{S}}_{\mathrm{M,D,PM}}$ is more expressive than $L^{\mathrm{A}}_{\mathrm{M,D,PM}}$. The scenario is similar to the previous one, but pattern matching lead us to consider a more complicated process, viz.

$$\langle a \rangle \mid (\ulcorner a \urcorner) \mid \langle b \rangle.\Omega \mid \langle a \rangle \mid \ldots \mid \langle a \rangle \mid \langle b \rangle \mid \ldots \mid \langle b \rangle$$

for a proper number of $\langle a \rangle$ and $\langle b \rangle$ in parallel. The idea is to prove that its encoding diverges. Roughly speaking, since the encoding of $\langle b \rangle.\Omega$ does not diverge, we have that the encoding

of Ω must be blocked by an input action relying on an actual template $\ulcorner m \urcorner$. We let the encoding of $\langle a \rangle \mid (\ulcorner a \urcorner)$ reduce until the moment in which $\langle m \rangle$ appears in the dataspace; then, we activate the encoding of $\langle b \rangle.\Omega$ that (possibly after some interactions with the encoding of $\langle a \rangle \mid \ldots \mid \langle a \rangle \mid \langle b \rangle \mid \ldots \mid \langle b \rangle$) yields a divergent process. See the full paper for further details.

$L^{\mathrm{S}}_{\mathrm{M,C,PM}}$ is more expressive than $L^{\mathrm{A}}_{\mathrm{M,C,PM}}$. Intuitively, communications in $L^{\mathrm{S}}_{\mathrm{M,C,PM}}$ atomically verify the channel and the sent value (if pattern matching is involved) and simultaneously activate the continuation of the sending process. Thus, $L^{\mathrm{A}}_{\mathrm{M,C,PM}}$ should provide the possibility of atomically verifying the channel and the sent value as well, but this excludes any information for synchronisation purposes. See the full paper for a formal proof.

6 Concluding Assessment

We have studied the impact of synchrony in the eight communication primitives that arise when combining three common and useful programming features: arity of data, communication medium and presence of pattern matching. Our results have been summarised in Figure 1; we now briefly discuss them.

It is evident that polyadicity is the only feature that alone ensures fully abstract encodings of synchrony in asynchrony: this is related to the possibility of equipping polyadic data exchanges with auxiliary information (either a restricted channel that will be exploited for acknowledgement purposes, or the length of the data) used to synchronise the sending and the receiving process.

For monadic and channel-based communications, we have that absence of pattern matching makes synchrony encodable asynchronously, whereas presence of pattern matching rules out any such (reasonable) encoding. The problem is that pattern matching introduces the possibility of atomically matching the name transmitted in the communication; this leaves no space for any auxiliary synchronisation information.

Finally, monadic and dataspace-based communications are too weak to ensure any reasonable encoding: the problem is that there is no way to associate a datum with the process that emitted it. The latter fact entails that those languages that exploit such primitives (e.g., Ambient [6] or CCS [14]) cannot freely interchange their synchronous and asynchronous versions, though the latter ones are still Turing powerful [6,3].

Acknowledgements. We would like to thank the EXPRESS'06 reviewers for their positive attitude and for several fruitful comments that improved the presentation of the work.

References

[1] G. Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA Sophia-Antipolis, May 1992.

[2] A. Brown, C. Laneve, and G. Meredith. πduce: a process calculus with native XML datatypes. In *Proc. of 2nd Int. Workshop on Services and Formal Methods*, volume 3670 of *LNCS*. Springer, 2005.

[3] N. Busi, R. Gorrieri, and G. Zavattaro. A process algebraic view of Linda coordination primitives. *Theoretical Computer Science*, 192(2):167–199, 1998.

[4] D. Cacciagrano and F. Corradini. On synchronous and asynchronous communication paradigms. In *Proc. of ICTCS'01*, number 2202 in LNCS, pages 256–268. Springer, 2001.

[5] D. Cacciagrano, F. Corradini, and C. Palamidessi. Separation of synchronous and asynchronous communication via testing. In *Proc. of EXPRESS*, ENTCS. Elsevier, 2005.

[6] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[7] G. Castagna, R. De Nicola, and D. Varacca. Semantic subtyping for the π-calculus. In *Proc. of LICS*, pages 92–101. IEEE Computer Society, 2005.

[8] R. De Nicola, D. Gorla, and R. Pugliese. On the expressive power of KLAIM-based calculi. *Theoretical Computer Science*, 356(3):387–421, 2006.

[9] R. De Nicola and M. Hennessy. Testing equivalence for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[10] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of POPL '96*, pages 372–385. ACM, Jan. 1996.

[11] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[12] D. Gorla. On the relative expressive power of asynchronous communication primitives. In *Proc. of FoSSaCS'06*, volume 3921 of *LNCS*, pages 47–62. Springer, 2006.

[13] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proc. of ECOOP '91*, volume 512 of *LNCS*, pages 133–147. Springer, 1991.

[14] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[15] R. Milner. The polyadic π-calculus: A tutorial. In *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993.

[16] R. Milner, J. Parrow, and J. Walker. A Calculus of Mobile Processes, I and II. *Information and Computation*, 100(1):1–40, 41–77, 1992.

[17] R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. of ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[18] V. Natarajan and R. Cleaveland. Divergence and fair testing. In *Proc. of ICALP'95*, volume 944 of *LNCS*, pages 648–659. Springer, 1995.

[19] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[20] J. Parrow. An introduction to the pi-calculus. In *Handbook of Process Algebra*, pages 479–543. Elsevier Science, 2001.

[21] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Foundations of Computing. MIT Press, May 2000.

[22] P. Quaglia and D. Walker. On synchronous and asynchronous mobile processes. In *Proceedings of FoSSaCS 2000*, volume 1784 of *LNCS*, pages 283–296. Springer, 2000.

[23] G. Tel. *Introduction to distributed algorithms*. Cambridge University Press, 1995.

Event Structure Spans for Non-deterministic Dataflow

Lucy Saunders-Evans[1,2]

Computer Laboratory
University of Cambridge
Cambridge, England

Glynn Winskel[3]

Computer Laboratory
University of Cambridge
Cambridge, England

Abstract

A compositional semantics for non-deterministic dataflow processes is described using spans of event structures; such a span describes a computation between datatypes, themselves represented by event structures, as itself an event structure. The spans of event structures represent certain profunctors (a generalisation of relations) used previously in a compositional semantics of non-deterministic dataflow and in the semantics of higher-order processes. Deterministic spans of event structures are shown to correspond to stable continuous functions and their semantics of dataflow to reduce to the usual fixed-point semantics of Kahn.

Keywords: Event Structures, Spans, Non-deterministic Dataflow.

1 Introduction

A dataflow process is built as a network of more basic processes connected by channels. In particular processes may be connected to allow feedback loops from output to input.

In [3], Kahn demonstrated that a denotational semantics could be given to a dataflow network of deterministic processes as a least-fixed point solution of a set of equations describing the components.

Brock and Ackerman showed in [1] that giving a semantics to nondeterministic dataflow processes was far from easy. In particular, *input-output* relations between sequences of values on input and output channels carry too little information about

[1] We thank the referees for their helpful suggestions.

[2] Email: ls275@cam.ac.uk

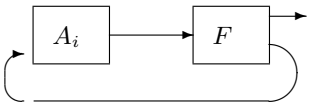
[3] Email: gw104@cam.ac.uk

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

the behaviour of networks to support a compositional semantics. The following example is essentially that presented in [9]. Let A_1 be the process that outputs a token, waits for a token on input and then outputs another token. Let A_2 be the process that either outputs a token and stops or waits for a token on input and then outputs two tokens. Let a token be represented by t. The input-output relations of A_1 and A_2 are the same:

$$\{(\emptyset,\ \emptyset),\ (\emptyset,\ t),\ (t,\ t),\ (t,\ tt)\}.$$

Let F be the process that copies every input to two outputs through which the output of the process A_i is fed back to the input channel. Observe that the process A_1 placed in this context produces a process which can output two tokens whereas the process A_2 results in a process that can only output a single token.

$C[A_i] =$ 

This confirms that there is no denotational semantics of nondeterministic dataflow in terms of traditional input-output relations. For similar reasons traditional uses of powerdomains also fall short.

Although traditional relations are insufficient, it was however shown in [2] that a form of relational compositional semantics was possible, but at the cost of moving to generalised relations, profunctors. Here we provide a representation of the *stable port profunctors* used there in terms of spans of event structures.

In more recent years feedback of the kind found in dataflow has reappeared in a variety of new contexts, which are condensed in a more abstract and general formulation of the properties a feedback operation, called *trace*, should satisfy. Let $\mathcal{C}$ be a category with a symmetric monoidal structure, $\otimes$. A trace operation for $\mathcal{C}$ is defined to be a family of operations, $Tr^C_{A,B} : \mathcal{C}(A \otimes C,\ B \otimes C) \to \mathcal{C}(A,\ B)$. The intuition behind this operation is illustrated in the following diagram:

A trace operation must obey a number of axioms to ensure it behaves correctly. The reader is referred to [2], which this paper supplements, for details and full references of this rich area.

2 Spans of Event Structures

Event structures model a process as a set of event occurrences with relations to express how events causally depend on others, or exclude other events from occurring.

Definition 2.1 An event structure is a tuple, $(E,\ \leq,\ \#)$, where:

- E is a set of events;
- $\leq$ is a partial order on E. Define $[e]$ to be $\{e' \mid e' \leq e\}$. It must be the case that

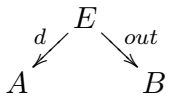
$[e] < \infty$ for all $e \in E$;

- $\#$ is a binary, irreflexive, symmetric relation such that $e_1 \# e_2$ and $e_2 \leq e_3$ implies that $e_1 \# e_3$.

Define the set of configurations (states) $\mathcal{C}(E)$ of an event structure E to be the downwards closed subsets of the set of events such that no two events are in conflict.

Definition 2.2 A set of configurations X of E is described as compatible iff $\bigcup X$ does not contain any conflicting events. We write $X \uparrow$ when X is compatible or in the case of two configurations w and z we write $w \uparrow z$.

It is well-known since [6] that event structures can represent both processes and datatypes, which at the time of [6] was remarked on as creating a curious mismatch with classical denotation semantics (where a processes denotes an element of a domain). The double role of event structures can be resolved by working with spans of event structures. A span



represents a computation process from a type, represented by event structure A, to the type B, an event structure, as again an event structure E. The morphisms d and *out* specify how the event structure inspects input and delivers output. There are many possible variations on spans as the morphisms d and *out* can have different natures. A systematic way to explore the range of possibilities via monads and distributed laws was suggested in [11] and is the subject of ongoing research. For the purposes of this paper however we can restrict to spans in which d is a demand morphism and *out* is rigid. Such spans first arose as representations of the profunctors used in a denotational semantics of higher-order processes [7] .

A *rigid* morphism, $f : E_1 \to E_2$, between event structures consists of a function between the respective sets of events that obeys the following properties:

- For all e_1 in E_1, $[f(e_1)] = f[e_1]$.
- For all e_1 and e_2 in E_1, if $f(e_1) = f(e_2)$ or $f(e_1) \# f(e_2)$ then $e_1 = e_2$ or $e_1 \# e_2$.

A rigid morphism determines a stable function on configurations (i.e., it preserves intersections of compatible configurations).

A *demand* morphism $d : E_1 \to E_2$ is a function from the set of events E_1 to the finite configurations of E_2. It must obey the following properties:

- $e_1 \leq e_2$ implies $d(e_1) \subseteq d(e_2)$;
- $d(e_1) \not\asymp d(e_2)$ implies $e_1 \# e_2$.

A demand morphism may be extended to act on configurations as well as events. Let $d : E_1 \to E_2$ be a demand morphism between E_1 and E_2. We can extend this to a function $d^\dagger : \mathcal{C}(E_1) \to \mathcal{C}(E_2)$ by

$$d^\dagger(x) \stackrel{def}{=} \bigcup_{e \in x} d(e)$$

for $x \in \mathcal{C}(E_1)$. The fact that $d^\dagger(x)$ is a configuration follows directly from x being

a configuration and that $d(e_1) \not\asymp d(e_2)$ implies that $e_1 \# e_2$.

For this paper, we restrict attention to spans of a special kind:

Definition 2.3 Let $A \stackrel{d}{\leftarrow} E \stackrel{out}{\rightarrow} B$ be a span of event structures where E, A and B are event structures, d is a demand morphism and out is a rigid morphism.

Where there is no confusion, we describe a span by its vertex i.e., $A \stackrel{d}{\leftarrow} E \stackrel{out}{\rightarrow} B$ will be referred to as E.

We can compose spans sequentially. Given two spans $A \stackrel{d_1}{\leftarrow} E_1 \stackrel{out_1}{\rightarrow} B$ and $B \stackrel{d_2}{\leftarrow} E_2 \stackrel{out_2}{\rightarrow} C$, their composition is the span $A \stackrel{d}{\leftarrow} E_3 \stackrel{out}{\rightarrow} C$ where E_3 is given as follows

- *Events*: $\{(x,\ e) \in \mathcal{C}(E_1) \times E_2 \mid g_1(x) = f_2(e)\}$;
- *Causality*: $(x_1,\ e_1) \leq (x_2,\ e_2)$ iff $x_1 \subseteq x_2$ and $e_1 \leq e_2$;
- *Conflict*: $(x_1,\ e_1) \# (x_2,\ e_2)$ iff $x_1 \not\asymp x_2$ or $e_1 \# e_2$.

The demand and output morphisms of the composition d and out are given by

$$d(x, e) \stackrel{def}{=} x \text{ and } out(x, e) \stackrel{def}{=} e\ .$$

The morphism d clearly has the properties of a demand morphism and out has the properties of a rigid morphism. This construction corresponds to a pullback construction in the category of event structures. (In fact spans form a bicategory in the usual manner of spans [5].)

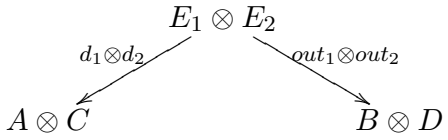
As well as sequential composition, it is also possible to define a parallel composition of spans (this forms the symmetric monoidal structure, $\otimes$, referred to in Section 1).

Definition 2.4 Given two event structures, $(E_1,\ \leq_1,\ \#_1)$ and $(E_2,\ \leq_2,\ \#_2)$, their parallel composition $(E_1,\ \leq_1,\ \#_1) \otimes (E_2,\ \leq_2,\ \#_2)$ is the event structure with:

- *Events*: $E_1 \biguplus E_2$
- *Causality*: $(i,\ e_1) \leq (j,\ e_2)$ iff $i = j$ and $e_1 \leq_i e_2$
- *Conflict*: $(i,\ e_1) \leq (j,\ e_2)$ iff $i = j$ and $e_1 \#_i e_2$

Where appropriate, we assume that the events in E_1 and E_2 are distinct and therefore write e_1 for $(i,\ e_1)$. We also make use of the notation $x \cap E_i$ for the projection $\pi_i(x)$ of a coonfiguration $x \in \mathcal{C}(E_1 \otimes E_2)$ to a configuration in E_i.

The parallel composition of $A \stackrel{d_1}{\leftarrow} E_1 \stackrel{out_1}{\rightarrow} B$ and $C \stackrel{d_2}{\leftarrow} E_2 \stackrel{out_2}{\rightarrow} D$ is



where $(d_1 \otimes d_2)(i,\ e) \stackrel{def}{=} \{i\} \times d_i(e)$ and $(out_1 \otimes out_2)(i,\ e) \stackrel{def}{=} (i,\ out_i(e))$ for $i = 1, 2$.

From these constructions, it is possible to give a semantics to non-deterministic dataflow.

3 Stable Families

It is sometimes difficult to define constructions on event structures directly and it is often simpler to first define them in terms of stable families, from which an event structure can then be obtained from the prime configurations. Stable families will be made use of in the definition of the trace construction.

Definition 3.1 A stable family, $\mathcal{F}$, is a set of sets with the following properties.

- *Coherence* $\forall X \subseteq \mathcal{F}.\ (\forall x,\ y \in X. x \uparrow y) \Rightarrow \bigcup X \in \mathcal{F}$.
- *Stability* $X \uparrow\Rightarrow \bigcap X \in \mathcal{F}$ for all non-empty $X \subseteq \mathcal{F}$. .
- *Coincidence-freeness*

  $\forall x \in \mathcal{F},\ e_1,\ e_2 \in x.\ e_1 \neq e_2 \Rightarrow$
  $\exists y \in \mathcal{F}.\ y \subseteq x\ \&\ ((e_1 \in y)\ \&\ (e_2 \notin y))\text{or}\ ((e_2 \in y)\ \&\ (e_1 \notin y))$.
- *Finiteness* $\forall x \in \mathcal{F}. \forall e \in x. \exists y \in \mathcal{F}.\ y \subseteq x\ \&\ e \in y\ \&\ |y| < \infty$.

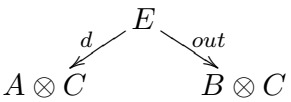
It can be deduced that a stable family, ordered by inclusion, forms a *prime algebraic* domain [6,10]. If x is a member of a stable family $\mathcal{F}$ and $e \in x$ let

$$[e]_x \stackrel{def}{=} \bigcap\{y \in \mathcal{F} | e \in y\ \&\ y \subseteq x\}.$$

Then $[e]_x \in \mathcal{F}$ and is called a prime configuration of $\mathcal{F}$. It is a complete prime of $(\mathcal{F},\ \subseteq)$. Because of coincidence-freeness, taking $max([e]_x) \stackrel{def}{=} e$ is well-defined.

4 Trace for Event Structures

This section is devoted to defining a trace operation $Tr^C_{A,B}$ which takes a span from $A \otimes C$ to $B \otimes C$ to a span from A to B. Consider a span



We first build a stable family out of those configurations of E which are *secure*. Roughly a configuration x is secure if the demand of each event $e \in x$ is met by the input in A or previous output to C, which we imagine is fed back as input. This idea is formalised below.

Definition 4.1 A configuration x of E is *secure* if each of its events is *secured* in x. An event, e, is secured in x iff

$$\exists e_1, \cdots,\ e_n \in x.\ e_n = e\ \&\ \forall i \leq n.\ \{e_1, ...,\ e_{i-1}\} \in \mathcal{C}(E)\ \ \&$$
$$d(e_i) \cap C\ \subseteq\ out\{e_1, ...,\ e_{i-1}\}.$$

Such a sequence, $e_1,\ ...,\ e_n$, is known as a securing sequence.

The subset of $\mathcal{C}(E)$, consisting of all the secure configurations will be shown to be a stable family, from which we then obtain an event structure. Our proofs will make use of a relation $\prec_x$ expressing the extra causal dependency on a configuration x of E which is introduced through feedback.

Definition 4.2 Let x be a configuration of E. For all events e_1 and e_2 in x, define

$e_1 \rightarrow_x e_2$ iff $out(e_1) \in d(e_2) \cap C$, and
$e_1 \prec_x e_2$ iff $e_1 < e_2$ or $e_1 \rightarrow_x e_2$.

We define $\{e\}_x$ to be the set ${\prec_x}^{\star\,-1}\{e\}$ for all $e \in x$.

Lemma 4.3 *An event e is secured in x iff*

i) $(d^\dagger(\{e\}_x)) \cap C \subseteq out(x)$ *and*

ii) $\prec_x$ *is well-founded on* $\{e\}_x$.

Proof. *if*: Assume (i) and (ii). In order to prove that event e is secured in x, we require a securing sequence for e. First note that the set $\{e\}_x$ is finite as $\prec_x$ is well-founded and the set of $\prec_x$-predecessors of an event is also finite. Thus we may tentatively take the securing sequence to be the set $\{e\}_x = \{e_1, \cdots, e_n\}$ with the events indexed such that

$$e_i \prec_x e_j \Rightarrow i < j \text{ and } e_n = e$$

Observe that the set $\{e_1, ..., e_{i-1}\}$ is a configuration of E for all $i \leq n$. This follows immediately from the definition of $\prec_x$.

It remains to confirm that, for all $i \leq n$, $d(e_i) \cap C \subseteq out\{e_1, ..., e_{i-1}\}$. Consider an event $c \in d(e_i) \cap C$ for some $i \leq n$. As $d^\dagger(\{e\}_x) \cap C \subseteq out(x)$, it is clear that $c = out(e_j)$ for some e_j in the sequence. It is the clearly the case that $e_j \prec_x e_i$ and therefore that $j < i$. This confirms that $c \in out\{e_1, ..., e_{i-1}\}$.

only if: Assume that event e is secured in configuration x of E and therefore that there is a securing sequence $e_1, ..., e_n$ such that $e_n = e$. Let $S = \{e_1, ..., e_n\}$.

We first show that

(†) $e' \prec_x e_i \Rightarrow e' \in \{e_1, ..., e_{i-1}\}$

and therefore that $e' = e_j$ for some $j < i$.

From the definition of $\prec_x$, if $e' \prec_x e_i$ then either $e' < e_i$ or $e' \rightarrow_x e_i$. As $\{e_1, ..., e_{i-1}\}$ is a configuration and therefore downwards closed with respect to $<$, if $e' < e_i$ then $e' \in \{e_1, ..., e_{i-1}\}$. If $e' \rightarrow_x e_i$ then $out(e') \in d(e_i) \cap C$. As out is a rigid morphism, $out(e'') = out(e''')$ implies that $e'' = e'''$ for all events, e'' and e''' in x. As $d(e_i) \cap C \subseteq out\{e_1, ..., e_{i-1}\}$, e' must therefore be a member of $\{e_1, ..., e_{i-1}\}$.

Observe that $d(e_i) \cap C \subseteq out\{e_1, ..., e_{i-1}\}$ for all i such that $1 \leq i \leq n$. It follows from (†) that $\{e\}_x \subseteq S$ and so $d^\dagger(\{e\}_x) \subseteq d^\dagger(S)$. As $d(e_i) \cap C \subseteq out\{e_1, ..., e_{i-1}\}$ for all $i \leq n$, $d^\dagger(S) \cap C \subseteq out\{e_1, ..., e_{n-1}\}$ and so $d^\dagger(S) \cap C \subseteq out(S)$. Recall that S is a subset of x and therefore property i must hold.

Property (†) implies that $\{e_i\}_x \subseteq \{e_1, ..., e_i\}$ and therefore that it is a finite set. If e_k is below e_l in a chain then $k < l$ so $\{e_k\}_x \subset \{e_l\}_x$. As $\{e_n\}_x$ is finite, there can therefore be no infinite chains. □

Corollary 4.4 *For all $x \in \mathcal{C}(E)$, x is secure iff $\prec_x$ is well-founded on x and $d^\dagger(x) \cap C \subseteq out(x)$.*

Proof. Follows directly from Lemma 4.3. □

In order to prove that the subset of $\mathcal{C}(E)$ consisting of the secure configurations is a stable family, we make use of the following Lemma:

Lemma 4.5 *Suppose x and y are secure configurations of E with $x \uparrow y$. Let $e \in x \cap y$. Then $e' \prec_x e$ iff $e' \prec_y e$ for all events e' and $\{e\}_x = \{e\}_y$.*

Proof. For all $e, e' \in x$, if $e' \prec_x e$ then either $e' < e$ or $e' \rightarrow_x e$. It is enough to prove that $e' \in y$. If the former is true then, as y is a configuration, $e \in y$ implies that $e' \in y$. If the latter then, as x and y are compatible, y is secure and $out(e') \in d(e) \cap C$, so $e' \in y$. □

Theorem 4.6 *The family consisting of all secure configurations of E is a stable family.*

Proof. Let $\mathcal{S} = \{x \in \mathcal{C}(E) \mid x \text{ secure}\}$. We show $\mathcal{S}$ is a stable family.

Coherence: $\forall X \subseteq \mathcal{S}.\ (\forall x,\ y \in X. x \uparrow y) \Rightarrow \bigcup X \in \mathcal{S}$.
Assume X is a pairwise compatible subset of $\mathcal{S}$. It is clear that $\bigcup X$ is a configuration of E and that, if $e \in \bigcup X$ then $e \in x$ for some x in X. As e is secured in x and $x \subseteq \bigcup X$, there is a securing sequence for e in $\bigcup X$.

Stability: $\forall X \subseteq \mathcal{S}.\ X \uparrow \Rightarrow \bigcap X \in \mathcal{S}$.
Suppose $X \subseteq \mathcal{S}$ and $X \uparrow$. Then as $\bigcap X \in \mathcal{C}(E)$ we only require in addition that $\bigcap X$ is secure. By Corollary 4.4 it suffices to show

(i) $\prec_{\bigcap X}$ is well-founded on $\bigcap X$, and
(ii) $d^\dagger(\bigcap X) \cap C \subseteq out(\bigcap X)$.

By Lemma 4.5, for $e \in \bigcap X$ we have that

$$\prec^\star_{\bigcap X}{}^{-1}\{e\} = \prec^{\star -1}_x \{e\}$$

for all $x \in X$. As each $\prec_x$ is well-founded on $x \in X$—each such x is secure—we obtain (i) that $\prec_{\bigcap X}$ is well-founded on $\bigcap X$. To show (ii) observe that

$$d^\dagger(\bigcap X) \cap C \subseteq d^\dagger(x) \cap C \subseteq out(x)$$

for all $x \in X$. Therefore

$$d^\dagger(\bigcap X) \cap C \subseteq \bigcap_{x \in X} out(x) = out(\bigcap X)$$

as *out* being rigid is a stable function.

Finiteness: $\forall x \in \mathcal{S}. \forall e \in x. \exists y \in \mathcal{S}.\ y \subseteq x$ and $e \in y$ and $|y| < \infty$.
If x is secure then each event e in x must have a securing sequence. This determines a finite secure configuration in $\mathcal{S}$ which contains e.

Coincidence-freeness: $\forall x \in \mathcal{S},\ e_1,\ e_2 \in x.\ e_1 \neq e_2 \Rightarrow$
$\exists y \in \mathcal{S}.\ y \subseteq x$ and $((e_1 \in y)$ and $(e_2 \notin y))$
or $((e_2 \in y)$ and $(e_1 \notin y))$.
Assume $e_1, e_2 \in x \in \mathcal{S}$ & $e_1 \neq e_2$.

Consider the secure configurations ${\prec_x^\star}^{-1}\{e_1\}$ and ${\prec_x^\star}^{-1}\{e_2\}$. If e_2 is a member of ${\prec_x^\star}^{-1}\{e_1\}$ then $e_2 \prec_x^+ e_1$. As x is secure, it cannot be the case that $e_1 \prec_x^+ e_2$. Therefore $e_1 \notin {\prec_x^\star}^{-1}\{e_2\}$ if $e_2 \in {\prec_x^\star}^{-1}\{e_1\}$ and vice versa. □

We can now define the trace operation.

Definition 4.7 Define $Tr(A \otimes C \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B \otimes C)$ to equal $A \overset{d'}{\leftarrow} E' \overset{out'}{\rightarrow} B$. E' is the event structure constructed as follows:

- *Events*: the prime configurations p of $\mathcal{S}$ for which $out(max(p)) \in B$.
- *Causality*: $p_1 \leq p_2$ iff $p_1 \subseteq p_2$.
- *Conflict*: $p_1 \# p_2$ iff $p_1 \not\asymp p_2$ in $\mathcal{S}$.

For $p \in E'$, we define $d'(p) \overset{def}{=} d^\dagger(p) \cap A$ and $out'(p) \overset{def}{=} out(max(p))$.

In order to show that out' is rigid we observe the following.

Lemma 4.8 *For e_1 and e_2 in $x \in \mathcal{C}(E)$, if $e_1 {\prec_x^\star} e_2$ and $e_1 \not\leq e_2$ then $out(e_1) \in C$.*

Proof. The above is proved by a simple induction on the length of the string of events between e_1 and e_2.

Suppose that $e_1 \prec_x e_2$. This implies that $e_1 \rightarrow_x e_2$ and so $out(e_1) \in C$. Suppose that the property holds for all strings of length less than or equal to k i.e., assume that if $e_1 {\prec_x}^k e_2$ and $e_1 \not\leq e_2$ then $out(e_1) \in C$. Suppose $e_1 {\prec_x}^{k+1} e_2$ and $e_1 \not\leq e_2$. Then there exists an e' with $e_1 {\prec_x}^k e'$ and $e' \prec_x e_2$. If $e_1 \not\leq e'$ then, by the induction hypothesis, $out(e_1) \in C$. If $e_1 \leq e'$ then, as $e_1 \not\leq e_2$, we have $e' \not\leq e_2$ and so $out(e') \in C$ as required. □

Lemma 4.9 *The map out' is a rigid morphism:*

Proof. From lemma 4.8, for p, p' in E', if $p \subseteq p'$ then $out(max(p)) \leq out(max(p')$ and so $out'(p) \leq out'(p')$. This implies that $out'[p] \subseteq [out'(p)]$ for all $p \in E'$. If $e \leq out(p')$ then, as there must be some element within p' that maps to e, it is clear that the downwards closed subset of p' with this as the maximum element will be mapped to e by out'. This implies that $[out'(p)] \subseteq out'[p]$.

Assume $out'(p') \# out'(p)$. This implies that $max(p') \# max(p)$ and therefore that $p' \# p$. □

Theorem 4.10 *$A \overset{d'}{\leftarrow} E' \overset{out'}{\rightarrow} B$ is a span.*

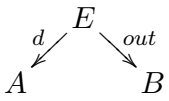
Proof. It is easily seen that E' is an event structure. Lemma 4.9 confirms that out' is indeed a rigid morphism. The fact that d' has the correct properties follows directly from the definition. □

Relying on previous work [2], we can show that the operation Tr obeys the trace axioms up to isomorphism. In [2], a trace operation was given for *stable port profunctors*. Spans of event structures represent such profunctors; a span $A \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B$ induces a profunctor $\hat{E}$ where $\hat{E}(a,\ b) \overset{def}{=} \{x \in \mathcal{C}(E) \mid d^\dagger(x) \subseteq a \ \&\ out(x) = b\}$. The representation respects both sequential and parallel composition as well as the trace operation.

5 Deterministic Dataflow

To connect with Kahn's and Plotkin's classic work [3,4] we define the notion of a *deterministic span*. These spans produce a unique maximum output for each input and therefore can be used to model deterministic dataflow processes. The trace construction described above is shown to correspond to the fixed point construction detailed in [3].

Definition 5.1 The span



is *deterministic* iff $d(e_1) \uparrow d(e_2) \Rightarrow \neg(e_1 \# e_2)$.

The extra requirement of the demand morphism ensures that no two conflicting events of E can require consistent configurations of A. Such spans can be related to functions between the configurations of A and B.

Definition 5.2 Let $A \stackrel{d}{\leftarrow} E \stackrel{out}{\rightarrow} B$ be a deterministic span. Define $\widetilde{E}\colon \mathcal{C}(A) \to \mathcal{C}(B)$ by

$$\widetilde{E}\,(a) \stackrel{def}{=} out\{e \in E \mid d(e) \subseteq a\}\ , \text{ for all } a \in \mathcal{C}(A).$$

Proposition 5.3 *The function $\widetilde{E}$ is stable and continuous for all deterministic spans E.*

Proof. As the demand of an event is a finite configuration it follows straightforwardly that $\widetilde{E}$ is continuous. To see it is stable we observe that $\widetilde{E}$ factors into the composition $out \circ F$ where

$$F(a) = \{e \in E \mid d(e) \subseteq a\}\ , \text{ for } a \in \mathcal{C}(A)\ .$$

The function $F : \mathcal{C}(A) \to \mathcal{C}(E)$ preserves all intersections, so certainly stable, while the function $out : \mathcal{C}(E) \to \mathcal{C}(B)$ is rigid so stable. Their composition $\widetilde{E}$ is therefore stable.

□

Proposition 5.4 *The composition of two deterministic spans is deterministic.*

Proof. Let $A \stackrel{d_1}{\leftarrow} E_1 \stackrel{out_1}{\rightarrow} B$ and $B \stackrel{d_2}{\leftarrow} E_2 \stackrel{out_2}{\rightarrow} C$ be deterministic spans.

Let $A \stackrel{d}{\leftarrow} E_3 \stackrel{out}{\rightarrow} C$ be their composition.

Let (x_1, e_1) and (x_2, e_2) be events in E_3 and assume $d(x_1, e_1) \uparrow\ d(x_2, e_2)$. This is true iff ${d_1}^{\dagger}(x_1) \uparrow {d_1}^{\dagger}(x_2)$. As E_1 is deterministic, $x_1 \uparrow x_2$. Consequently, $d_2(e_1) \uparrow d_2(e_2)$ from the definition of composition. This implies that $\neg(e_1 \# e_2)$, as E_2 is deterministic. As $x_1 \uparrow x_2$ and $\neg(e_1 \# e_2)$ it follows that $\neg((x_1, e_1) \# (x_2, e_2))$. □

Lemma 5.5 *Let $A \otimes C \stackrel{d}{\leftarrow} E \stackrel{out}{\rightarrow} B \otimes C$ be a deterministic span.*

Let $a \in \mathcal{C}(A)$. There is a continuous function $\varphi : \mathcal{C}(E) \to \mathcal{C}(E)$ such that $Sec(a) \stackrel{def}{=} \mu S.\varphi(S)$, the least fixed point of φ, is a secure configuration of E and

moreover is the maximum secure configuration x of E such that

$$d^\dagger(x) \cap A \subseteq a.$$

Proof. For $S \subseteq E$ define

$$\varphi(S) = \{e \in E \mid d(e) \subseteq a \cup (C \cap out(S))\}.$$

That φ is a function between configurations follows from determinism, while continuity follows from finiteness of demands as in the proof of Proposition 5.3.

Then $\mu S.\varphi(S) = \bigcup_{i\in\omega} S_i$, a union of an $\subseteq$-increasing chain of configurations of E given inductively by

$$S_0 = \emptyset$$
$$S_{i+1} = \varphi(S_i).$$

Observe that, as φ is a function from configuration to configuration, $S_i \in \mathcal{C}(E)$ for all i.

We show by induction that S_i secure for all $i \in \omega$.

Base case: The property trivially holds for S_0.

Inductive step: Assume that S_k is secure. Each event in S_k is secured and therefore has a securing sequence within S_k. Any additional events e in S_{k+1} have $d(e) \cap C \subseteq out(S_k)$. We can therefore give a securing sequence for e by combining the securing sequences of the events upon which it depends according to $\prec_{S_{k+1}}$. Therefore S_{k+1} is secure. This shows that $Sec(a)$ is secure for all $a \in \mathcal{C}$ and it is clear that $d^\dagger(Sec(a)) \cap A \subseteq a$.

Suppose there exists an $x \in \mathcal{C}(E)$ that is secure and for which $d^\dagger(x) \cap A \subseteq a$.

We show by well-founded induction on $\prec_x$ that

$$\forall e \in x.\ e \in Sec(a).$$

Suppose $e' \in Sec(a)$ for all $e' \prec_x e$. Then $d(e) \cap C \subseteq out(Sec(a))$ as x is secure and by assumption $d(e) \cap A \subseteq a$. Hence $d(e) \subseteq a \cup (C \cap out(Sec(a)))$ and therefore $e \in \varphi(Sec(a)) = Sec(a)$. □

Corollary 5.6 *If a span is deterministic then its trace is deterministic.*

Proof. Let $A \otimes C \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B \otimes C$ be a deterministic span. Let $p_1, p_2 \in Tr(E)$ with demands $a_1,\ a_2 \in \mathcal{C}(A)$ such that $a_1 \uparrow a_2$.

As each $d^\dagger(p_i) \subseteq a_i \cup (C \cap out(p_i))$ for $i = 1,\ 2$, they are also both subsets of $a_1 \cup a_2 \cup (C \cap out(p_i))$. Hence as p_1 and p_2 are secure $p_1,\ p_2 \subseteq Sec(a_1 \cup a_2)$ and so $p_1 \uparrow p_2$. □

Let $A \overset{dem}{\leftarrow} E \overset{out}{\rightarrow} B$ be a deterministic span.

Lemma 5.7 *For all $w \in \mathcal{C}(A)$,*

$$\widetilde{Tr(E)}\ (w) = B \cap outSec(w)\ .$$

Proof. By definition

$$\widetilde{Tr(E)}\ (w) = out\{p \in Tr(E) | A \cap d^\dagger p \subseteq w\}\ .$$

If $A \cap d^\dagger p \subseteq w$ with $p \in Tr(E)$ then p is secure, so $p \subseteq Sec(x)$ by the maximum property of $Sec(x)$—Lemma 5.5. It follows that $\widetilde{Tr(E)}\ (w) \subseteq B \cap out(Sec(w))$. To see the reverse inclusion, note that for any $e \in Sec(w)$ with $out(e) \in B$ we have $[e]_{Sec(w)} \in Tr(E)$. □

In conclusion we can understand the trace of a deterministic span in terms of a least fixed point of its associated function; the trace of deterministic event structures reduces to the trace known to Kahn [3].

Theorem 5.8 *Let $A \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B$ be a deterministic span. For all $w \in \mathcal{C}(A)$,*

$$\widetilde{Tr(E)}\ (w) = B \cap \mu z \in \mathcal{C}(B \otimes C).\ \widetilde{E}\ (w \cup (C \cap out(z)))\ .$$

Proof. By Lemma 5.7,

$$\widetilde{Tr(E)}\ (w) = B \cap out(\mu x.\, \phi(x))$$

where $\phi : \mathcal{C}(E) \to \mathcal{C}(E)$ is the continuous function given by

$$\phi(x) = \{e \in E | d(e) \subseteq w \cup (C \cap out(x))\}$$

for $x \in \mathcal{C}(E)$.

Define the continuous function $\psi : \mathcal{C}(B \otimes C) \to \mathcal{C}(B \otimes C)$ by taking

$$\psi(u) = \widetilde{E}\ (w \cup (C \cap u))$$

for $u \in \mathcal{C}(B \otimes C)$.

It is easy to see that $out : E \to \mathcal{C}(B \otimes C)$ is a strict continuous function between the domains of configurations. Directly from their definition we see that

$$out \circ \phi(x) = \psi \circ out(x)$$

for $x \in \mathcal{C}(E)$; both expressions equal

$$out\{e \in E | d(e) \subseteq w \cup (C \cap outx)\}\ .$$

By a well-known property of least fixed points we now know that $\mu u.\, \psi(u) = out(\mu x.\, \phi(x))$, which gives the result directly. □

6 Concluding Remarks

Event structure spans provide an intuitive semantics for both deterministic and nondeterministic dataflow processes. This semantics is a good example of the way in which event structures can be used to represent both types and operations between types.

The spans of event structures used here were first discovered in joint work with Mikkel Nygaard [7] where they appeared as an informative, more operational representation of the profunctors used as denotations of an affine language for higher-order processes. It was realised later that they also represented the profunctors

used in an earlier 'relational' model of nondeterministic dataflow [2]. Such spans are part of a more general picture, the beginnings of which are sketched in [11]. But we believe that our treatment of nondeterministic dataflow in itself makes a convincing case for the usefulness of spans of event structures in semantics.

References

[1] Brock, J. and W. Ackerman, *Scenarios: A model of non-determinate computation*, Formalization of programming concepts **107** (1981).

[2] Hildebrandt, T. T., P. Panangaden and G. Winskel, *A relational model of nondeterministic dataflow*, MSCS (2003).

[3] Kahn, G., *The semantics of a simple language for parallel programming*, Information Processing **74** (1974), pp. 471–475.

[4] Kahn, G. and G. Plotkin, *Concrete domains*, Theoretical Computer Science **121** (1993), pp. 197–277.

[5] MacLane, S., "Categories for the Working Mathematician," Graduate Texts in Mathematics **5**, Springer, 1998, second edition.

[6] Nielson, M., G. D. Plotkin and G. Winskel, *Petri nets, event structures and domains*, Theoretical Computer Science **13(1)** (1981), pp. 85–108.

[7] Nygaard, M., "Domain Theory for Concurrency," Ph.D. thesis, University of Aarhus (2003).

[8] Rabinovich, A. and B. A. Trakhtenbrot, *Communication among relations*, in: M. Paterson, editor, *Proceedings of the 6th ICALP*, LNCS **443** (1990), pp. 294–307.

[9] Russell, J. R., *Full abstraction for nondeterministic dataflow networks*, in: *FOCS*, 1989, pp. 170–175.

[10] Winskel, G., *Event structure semantics for CCS and related languages*, in: *ICALP '82*, 1982.

[11] Winskel, G., *Relations in concurrency (invited talk)*, LICS'05 (2005), full version available at www.cl.cam.ac.uk/users/gw104/.

Extensionality of Spatial Observations in Distributed Systems

Luís Caires [1] and Hugo Torres Vieira [2]

CITI / Departamento de Informática, FCT Universidade Nova de Lisboa, Portugal

Abstract

We discuss the tensions between intensionality and extensionality of spatial observations in distributed systems, showing that there are natural models where extensional observational equivalences may be characterized by spatial logics, including the composition and void operators. Our results support the claim that spatial observations do not need to be always considered intensional, even if expressive enough to talk about the structure of systems. For simplicity, our technical development is based on a minimalist process calculus, that already captures the main features of distributed systems, namely local synchronous communication, local computation, asynchronous remote communication, and partial failures.

Introduction

Logical characterizations of concurrent behaviors have been introduced for a long time now. A fundamental result in the field, due to Hennessy and Milner [13], is the characterization of behavioral equivalence in process algebras as indistinguishability with respect to a modal logic. Such results are important not only theoretically, but also because of their influence in the design of practical specification languages for software systems. Hennessy-Milner logic (HML) adds to propositional operators the action modality $\langle\lambda\rangle A$, allowing the logic to observe a grain of behavior: a process satisfies $\langle\lambda\rangle A$ if it satisfies A after performing action λ. HML characterizes behavioral equivalence in the sense that two processes are strongly bisimilar if and only if they satisfy exactly the same formulas.

More recently, spatial logics for concurrency [6,9,4] have been proposed with the aim of specifying distributed behavior and other essential aspects of distributed computing systems. In general terms, these developments reflect a shift of focus in concurrency research, that has been building up from the last decade on, from the study of centralized concurrent systems to the study of general distributed systems. While centralized processes may be accurately modeled as pure objects of behavior, in distributed systems many interesting

[1] Luis.Caires@di.fct.unl.pt
[2] htv@di.fct.unl.pt

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

phenomena besides pure interaction, such as location dependent behavior, resource usage, and mobility, must be considered.

Present in all spatial logics for concurrency are the composition operator $A \mid B$ and the void operator $\mathbf{0}$ [4]. Intuitively, a system satisfies $A \mid B$ if it can be decomposed in two disjoint subsystems such that one satisfies A and the other satisfies B, while a system satisfies $\mathbf{0}$ if it is the empty system. The guarantee (logical adjunct of the composition operator) $A \triangleright B$, introduced in [9], allows the logic to talk about contextual properties. Namely, a process satisfies $A \triangleright B$ if whenever composed with a system that satisfies A, yields a (possibly larger) system that satisfies B. Decomposition and composition of systems as mentioned here is generally interpreted up to structural congruence, and thus structural congruence seems to play a key role in the semantics of spatial logics.

Observation of features such as spatial separation are frequently considered intensional because they usually induce fine distinctions among processes that are not substantiated by purely behavioral (extensional) observations. According to Sangiorgi [20], "A logic is intensional if it can separate terms on the basis of their internal structure, even though their behaviors are the same". Moreover, in many situations, it turns out that the logical equivalence induced by a spatial logic on processes, is not only strictly finer than behavioral congruence, but coincides with structural congruence [20,5,11,21].

These results contributed to widespread the impression that spatial observations, as those induced by spatial connectives, are intrinsically intensional, imposed extraneously so to increase the power of the observer. For example, Hirschkoff has shown [14] that if the so-called intensional connectives composition and void are removed from a spatial logic for the pi-calculus, while retaining the guarantee, one obtains a logic whose separation power precisely coincides with strong bisimulation and may then be considered extensional. The ability of the spatial connectives to capture structural congruence is also attributed to their ability to count, separate, and express arithmetical constraints, *e.g.*, about the number of subsystems of a given system. The observational power of spatial logics may then sometimes appear a bit arbitrary, in the sense that structural congruence does not have a canonical status among behavioral process equivalences, and is frequently seen just as a technical convenience, with a syntactic flavor, to ease the presentation of a calculus operational semantics.

On the other hand, it has been argued [4,2,3] that the intensional character of logical characterizations of spatiality in distributed computation may be, at least in part, incidental, and does not necessarily reflect the fundamental motivation for introducing spatial logics for concurrency. Ideally, we would like spatial observations, as captured by spatial logics, to reflect natural distinctions and similarities between distributed systems, in a context where spatial location is a relevant observable, in parity with more standard behavioral observables. We expect spatial observations of the sort, captured by spatial logic operators such as composition, to be taken modulo an intended notion of equality of the observable space-time structure, independently on whether such equality relation is technically defined using a notion of structural congruence. If certain spatial-behavioral observations precisely capture the observable structure of a model in our sense, they would have to be considered extensional, even if able to detect aspects of spatial structure.

In this paper, we pursue the informal discussion started above in technical terms. Namely, we make precise the claim that spatial observations, including structural ones, may be understood as purely extensional in fairly natural models of distributed systems. To discuss

the several issues of interest in a simplified setting, we consider a minimal distributed process calculus, obtained by extending the smallest concurrent fragment of CCS with flat anonymous locations. Our model can be seen as a general abstraction of the essence of distributed systems, already featuring all the key ingredients present in distributed process calculi, although in a possibly less refined way. Processes may synchronously communicate locally to a site through standard CCS-like synchronization, and asynchronously communicate at a distance, by means of a migration primitive. We also allow systems to nondeterministically exhibit partial failures, as in [1,12]. Notice that it is not our aim here to propose yet another distributed process calculus, but rather to set up a convenient setting to compare distributed system observational equivalences and their spatial logical characterizations.

Our technical contributions may be summarized as follows. After introducing the process calculus and its reduction semantics, we define observational equivalence by adopting the canonical notion of reduction barbed congruence. Barbed congruence [17] and reduction barbed congruence [16] are currently accepted as the standard approach to define reference behavioral equivalences for general process calculi. After showing some basic properties of reduction barbed congruence in our setting, we define strong bisimulation, an alternative coinductive characterization of observational equivalence, which is shown equivalent to reduction barbed congruence. The interesting aspect of our definition of strong bisimulation is that it contains "intensional" clauses (in the sense of [20]), namely a clause expressing separation, and a clause for observing the empty system. We then use the characterization of reduction barbed congruence in terms of strong bisimulation to identify a spatial logic characterization of both reduction barbed congruence and strong bisimulation: our logic is an extension of HML with the composition and void operators of spatial logic. The same line of development is also carried out for the weak case. In this latter setting, we prove minimality of the logic, thus showing the essential role of all of the logic operators, in particular of the spatial operators, in the intended expressive and separation power. We can verify that in both the strong and weak cases the process equivalences induced by the logics are coarser than structural congruence, and that the presence of the composition and void operators, semantically interpreted in the standard way, do not carry any lack of extensionality (with extensionality interpreted with reference to a standard observational equivalence), even if the logics can express separation and counting constraints on the structure of systems.

1 A Simple Model of Distributed Systems

In this section we present the syntax and operational semantics of our distributed process calculus. Assume given an infinite set Λ of *names*, ranged over by a, b, c.

Definition 1.1 [Actions, Processes and Networks] The sets $\mathcal{A}$ of *actions*, $\mathcal{P}$ of *processes*, and $\mathcal{N}$ of *networks* are given by:

$$\alpha ::= \bar{a} \mid a \mid \tau \qquad P, Q ::= \mathbf{nil} \mid P \mid Q \mid \alpha.P \mid \mathbf{go}.P \qquad N, M ::= \mathbf{0} \mid N \mid M \mid [P]$$

For actions we consider the output $\bar{a}$, the input a and the internal computation τ. For processes, we consider the smallest fragment of CCS featuring some form of concurrency, thus we have inaction **nil**, parallel composition $P \mid Q$, and action prefixing $\alpha.P$. On top

of this, we introduce a notion of distribution by locating processes P inside sites of the form $[P]$, anonymous for simplicity, and by adding the migration capability $\mathbf{go}.P$ to processes which, since sites are not natively named, allows processes to non-deterministically migrate to other sites. A distributed system is thus represented by a network consisting of a collection of sites spread in space, by means of spatial composition $N \mid M$, which we will abbreviate using $\prod_{j\in J}\left[P^j\right]$ for a J-fold collection of sites. $\mathbf{0}$ stands for the empty network. We use $\mathit{fn}(N)$ to denote the set of free names of a network N, defined as usual. The operational semantics of our calculus follows, captured by the relations of structural congruence and reduction.

Definition 1.2 [Structural congruence] *Structural congruence*, noted $\equiv$, is the least congruence on processes and networks such that $(\mathcal{P}, \mathbf{nil}, |)$ and $(\mathcal{N}, \mathbf{0}, |)$ are commutative monoids, and $P \equiv Q$ implies $[P] \equiv [Q]$.

Definition 1.3 [Reduction] *Reduction*, noted $N \rightarrow M$, is the relation between processes inductively defined as follows

$$[\bar{a}.P \mid a.Q \mid R] \rightarrow [P \mid Q \mid R] \text{ (Red Comm)} \qquad [\tau.P \mid Q] \rightarrow [P \mid Q] \text{ (Red Tau)}$$

$$[\mathbf{go}.P \mid Q] \mid [R] \rightarrow [Q] \mid [P \mid R] \text{ (Red Go)} \qquad [P] \mid N \rightarrow \mathbf{0} \text{ (Red Fail)}$$

$$\frac{N \rightarrow N'}{N \mid M \rightarrow N' \mid M} \text{ (Red Cong)} \qquad \frac{N \equiv N' \rightarrow M' \equiv M}{N \rightarrow M} \text{ (Red Struct)}$$

The rule (Red Comm) specifies interaction between two processes through co-action synchronization locally inside a site, while rule (Red Tau) specifies internal action of a process. Rule (Red Go) specifies that a process prefixed by **go** may migrate to another site. Rule (Red Fail) expresses that any non-empty network may fail, thus modeling fail-stop failure of an arbitrary subsystem.

Our aim now is to define a natural notion of observational equivalence on networks. To that end, we adopt the canonical notion of reduction barbed congruence, according to which two systems are observationally equivalent if no context can distinguish between them by barb detection. In our case, we restrict to one-hole spatial contexts, as *e.g.*, in [1,12], hence of the form $C\,[\bullet] ::= N \mid \bullet$, for some network N.

We use the standard notion of barb observation [17], even if it assumes in a sense the existence of a global observer, which might be debatable in the context of distributed systems. Thus a network N exhibits barb a, noted $N{\downarrow_a}$, if there are P, Q, M such that $N \equiv [a.P \mid Q] \mid M$, hence reflecting the fact that any external observer can get to know that an input is ready via some channel name, at some accessible site. We now define our reference observational equivalence relation, along the lines of [18,16].

Definition 1.4 [Strong reduction barbed congruence] *Strong reduction barbed congruence*, noted $\simeq$, is the largest symmetric relation R such that for all $(N, M) \in R$:

For all barbs a, if $N{\downarrow_a}$ then $M{\downarrow_a}$	(Barb closed)
If $N \rightarrow N'$ then there is M' s.t. $M \rightarrow M'$ and $(N', M') \in R$	(Reduction closed)
For all contexts $C\,[\bullet]$, $(C\,[N]\,, C\,[M]) \in R$	(Context closed)

We establish some standard properties of strong reduction barbed congruence, such as

$\simeq$ is a congruence. Notice that we just consider in this paper, congruences under spatial (static) contexts. As explained above, this does not carry a lack of generality, given the main motivations of our development. Moreover:

Proposition 1.5 *We have* $\equiv \subset \simeq$.

Proof. The proof of $\subseteq$ follows standard lines. To prove that $\equiv$ is strictly included in $\simeq$ we may show that $[a.\mathbf{nil} \mid a.\mathbf{nil}] \simeq [a.a.\mathbf{nil}]$ but $[a.\mathbf{nil} \mid a.\mathbf{nil}] \not\equiv [a.a.\mathbf{nil}]$. □

It follows from the congruence property that strong reduction barbed congruence is closed under composition. In particular for site composition, we have:

Lemma 1.6 *Let* P^i *and* Q^i $(i \in J)$ *be collections of processes. If for all* $i \in J$ *we have* $\big[P^i\big] \simeq \big[Q^i\big]$, *then also* $\prod_{j\in J}\big[P^j\big] \simeq \prod_{j\in J}\big[Q^j\big]$.

Although Definition 1.4 is standard, with reference to the global observation of barbs in networks, observations already leak some relevant information about the distributed structure of systems. Lemma 1.7 states that strong reduction barbed congruent networks always result from an underlying one-one and onto correspondence of strong reduction barbed congruent sites. In particular, we conclude strong reduction barbed congruent networks always have the same number of sites.

Lemma 1.7 *Let* M, N *be networks such that* $N \triangleq \prod_{j\in J}\big[P^j\big]$, *where* P^j $(j \in J)$ *is a collection of processes, and* $N \simeq M$. *Then there is a collection of processes* Q^j $(j \in J)$ *such that* $M \equiv \prod_{j\in J}\big[Q^j\big]$ *and for all* $j \in J$ *we have* $\big[P^j\big] \simeq \big[Q^j\big]$.

Proof. (Sketch, full proof in [7]) We consider a context that holds processes that may migrate and *mark* every site of N with an input on the unique name, and we make sure that every input is located at a different site. Since M behaves the same as N under this context (and using a symmetric reasoning) we obtain that M has $\#J$ sites. We then exploit failures in N that leave only a single site active, being that this behavior must be mimicked by failures in M that also leave just one site up. These singled out sites are strong reduction barbed congruent, hence hold the same unique input, thus ensuring an unique correspondence. We then consider another context that may clean up the marker and all other foreign elements, which then allows us to conclude the sites were originally strong reduction barbed congruent. □

2 Strong Bisimilarity

Since strong reduction barbed congruence relies on universal quantification over all contexts, we now propose a more manageable characterization of observational equivalence. More concretely, we introduce a labeled transition system with the aim of capturing the contextual behavior of the networks, by means of observing process commitments, in turn expressed by transition labels. Building on such labeled transition system, a coinductive definition of bisimilarity is then presented.

The set of transition labels, noted $\mathcal{L}$, is given by $\mathcal{L} \triangleq \{\alpha \mid \alpha \in \mathcal{A}\} \cup \{[a] \mid a \in \Lambda\}$, and ranged over by λ. Transition labels reflect internal computation (τ), and abstract communication and mobility from and to the external environment. Output ($\bar{a}$) and input (a) transitions represent the interaction with a process that migrates from the outer environment and communicates on a given channel.

Grow transitions ($[a]$) are used to allow the observation of process migration from the system to the external environment. Such a grow transition allows the labeled transition system to import a minimal piece of the external environment, providing the system with a candidate migration target. This turns out to be essential for covering the case of networks with a single site, since in that case only the enlargement of the system with a new site gives processes intending to migrate a possible destination.

Given these ingredients, we define our labeled transition system. As we will show later, these labels are essential to capture reduction barbed congruence.

Definition 2.1 [Commitment] *Commitment*, noted $N \xrightarrow{\lambda} M$, is the relation on processes and labels inductively defined as follows

$$[\bar{a}.P \mid a.Q \mid R] \xrightarrow{\tau} [P \mid Q \mid R] \text{ (Comm)} \qquad [\tau.P \mid Q] \xrightarrow{\tau} [P \mid Q] \text{ (Tau)}$$

$$[\bar{a}.P \mid Q] \xrightarrow{\bar{a}} [P \mid Q] \text{ (Out)} \qquad [a.P \mid Q] \xrightarrow{a} [P \mid Q] \text{ (In)}$$

$$[\mathbf{go}.P \mid Q] \mid [R] \xrightarrow{\tau} [Q] \mid [P \mid R] \text{ (Go)}$$

$$[P] \mid N \xrightarrow{\tau} \mathbf{0} \text{ (Fail)} \qquad N \xrightarrow{[a]} N \mid [a.\mathbf{nil}] \text{ (Grow)}$$

$$\frac{N \xrightarrow{\lambda} N'}{N \mid M \xrightarrow{\lambda} N' \mid M} \text{ (Cong)} \qquad \frac{N \equiv N' \xrightarrow{\lambda} M' \equiv M}{N \xrightarrow{\lambda} M} \text{ (Struct)}$$

We can verify that τ commitments match reductions and conversely. Notice that although *e.g.*, the systems $[\mathbf{nil}] \mid [\mathbf{nil}]$ and $[\tau.\mathbf{nil}]$ have exactly the same commitment graph, they are not observationally equivalent in the light of Lemma 1.7. Thus, in order to properly capture strong reduction barbed congruence, we include in the definition of strong bisimulation two spatial clauses (referred to as "intensional clauses" in [20]).

We then have:

Definition 2.2 [Strong Bisimulation] A binary relation $B \subseteq \mathcal{N} \times \mathcal{N}$ is a *strong bisimulation* if and only if it is symmetric and whenever $(N, M) \in B$ then

$$N \equiv N' \mid N'' \Rightarrow \exists M', M'' \,.\, M \equiv M' \mid M'' \wedge (N', M') \in B \wedge (N'', M'') \in B$$

$$N \equiv \mathbf{0} \Rightarrow M \equiv \mathbf{0}$$

$$N \xrightarrow{\lambda} N' \Rightarrow \exists M' \,.\, M \xrightarrow{\lambda} M' \wedge (N', M') \in B$$

We remark that the second clause in Definition 2.2 is subsumed by the third one since only void systems have no possible internal actions (due to failures), however we prefer to include it in the definition for the sake of uniformity with the corresponding weak version, and thus avoid some extra incidentality.

Notice also that the first clause properly distinguishes $[\tau.\mathbf{nil}]$ and $[\mathbf{nil}] \mid [\mathbf{nil}]$, because there is no way to split $[\tau.\mathbf{nil}]$ (up to $\equiv$) in two parts with some transition each.

We prove that strong bisimulations are equivalence relations closed under union, and define:

Definition 2.3 [Strong bisimilarity] *Strong bisimilarity*, noted $\sim$, is the largest strong bisimulation.

2.1 Full Abstraction

This section is devoted to proving that strong bisimilarity, as defined in Definition 2.3, characterizes strong reduction barbed congruence in a fully abstract way. The proof builds on a series of intermediate technical results.

Lemma 2.4 *Let M be a network and $P^j (j \in J)$ a collection of processes where $\prod_{j\in J}\left[P^j\right] \sim M$. Then there is a collection of processes $Q^j (j \in J)$ such that $M \equiv \prod_{j\in J}\left[Q^j\right]$ and for all $j \in J$, $\left[P^j\right] \sim \left[Q^j\right]$.*

Proof. By induction on the size of J, using the separation and emptiness clauses. □

The proof of the main result of this section (Theorem 2.6) is not technically involved, but critically depends on next Lemma 2.5, that expresses a key compositionality principle of our calculus. Notice that the basic building block of systems referred to in the statement of Lemma 2.5 is the process: since we have to take migration into account, it is essential to assure compositionality at the process level. We abbreviate collections of sites such that each one holds a collection of processes.

Lemma 2.5 *Let J be a finite set and I_j, for all $j \in J$, be a finite set. Let P_i^j and Q_i^j be processes such that for all $j \in J$ and $i \in I_j$ we have $\left[P_i^j\right] \sim \left[Q_i^j\right]$. Then*

$$\prod_{j\in J}\left[\prod_{i\in I_j} P_i^j\right] \sim \prod_{j\in J}\left[\prod_{i\in I_j} Q_i^j\right]$$

Proof. (Sketch, full proof in [7]) By coinduction on the definition of strong bisimulation. We sketch the proof for the interesting case of migration.

We exploit the grow transition using a fresh name, in the sense that it does not occur in neither one of the P_i^js and Q_i^js, which creates a possible target for migrations and allows us to isolate migrating processes, since we can decompose and observe the input on the fresh name. Using this technique and since we can establish that the newly created sites are bisimilar, we can be sure to obtain a collection of sites that respects the statement of the Lemma for any choice of target of the migration. Notice that a migration on one side need not be always matched by a migration on the other, because the migrating process can *e.g.*, be inaction, in which case, a migration may be matched by an internal computation step. □

By Lemma 2.4 and Lemma 2.5 we prove strong bisimilarity is a congruence, from which follows, in standard lines, that $\sim \subseteq \simeq$. We then prove $\simeq \subseteq \sim$, using Lemma 1.6 and Lemma 1.7 to address the structural issues. We can then state:

Theorem 2.6 (Full abstraction) *We have $\sim = \simeq$.*

2.2 Logical Characterization of Strong Bisimilarity

In this section, we characterize strong bisimilarity (and thus strong reduction barbed congruence) in logical terms, using a simple spatial logic.

Definition 2.7 [Spatial logic $\mathcal{L}_s$] Formulas are defined by the following syntax:

$$\text{(Formulas) } A, B, C ::= \mathbf{T} \;\Big|\; \neg A \;\Big|\; A \wedge B \;\Big|\; \mathbf{0} \;\Big|\; A \mid B \;\Big|\; \langle\lambda\rangle A$$

Our logic, besides the usual action modality from HML, includes the composition and void operators of spatial logics, interpreted in the standard way. For example, we may express property "network has exactly one site" by the formula $\neg\mathbf{0} \wedge \neg(\neg\mathbf{0} \mid \neg\mathbf{0})$. The semantics of the logic is given by the denotation of the formulas, i.e., a formula denotes the set of networks that satisfy it.

Definition 2.8 [Semantics of $\mathcal{L}_s$] A formula's denotation is inductively given by

$$[\![\mathbf{T}]\!] \triangleq \mathcal{N} \quad [\![\neg A]\!] \triangleq \mathcal{N} \backslash [\![A]\!] \quad [\![A \wedge B]\!] \triangleq [\![A]\!] \cap [\![B]\!] \quad [\![\mathbf{0}]\!] \triangleq \{N \mid N \equiv \mathbf{0}\}$$
$$[\![A \mid B]\!] \triangleq \{N \mid \exists N', N'' \,.\, N \equiv N' \mid N'' \wedge N' \in [\![A]\!] \wedge N'' \in [\![B]\!]\}$$
$$[\![\langle\lambda\rangle A]\!] \triangleq \{N \mid \exists N' \,.\, N \xrightarrow{\lambda} N' \wedge N' \in [\![A]\!]\}$$

We write $N \models A$ to mean $N \in [\![A]\!]$. We say that networks M and N are *logically equivalent* w.r.t. $\mathcal{L}_s$, written $M =_{\mathcal{L}_s} N$, if and only if they satisfy exactly the same formulas of $\mathcal{L}_s$, namely if and only if, for any formula A of $\mathcal{L}_s$, we have $M \models A \iff N \models A$. We now state our logical characterization result.

Theorem 2.9 (Logical Characterization of $\sim$) *We have $\sim = =_{\mathcal{L}_s}$.*

Proof. (Sketch, full proof in [7]) Proof of $\sim \subseteq =_{\mathcal{L}_s}$ follows by a standard induction on the structure of the formulas. We prove $=_{\mathcal{L}_s} \subseteq \sim$ by coinduction on the definition of strong bisimulation, using the witness $R \triangleq \{(N, M) \mid N =_{\mathcal{L}_s} M\}$. Proof of the emptiness clause is immediate. For both the separation and transition clauses we build on the fact that the image set of the transition for the latter and of all possible decompositions for the former is finite (up to structural congruence). We then exploit the finiteness of these finite sets to prove that there is a (logical equivalent) correspondence between at least one of their elements. Otherwise we could collect the finite set of all formulas that distinguish them in a conjunction that must hold for both networks, either after a decomposition or after an action, since they are logically equivalent. We then obtain our bisimilar result by coinduction. □

As a corollary we immediately conclude that $=_{\mathcal{L}_s}$ precisely characterizes $\simeq$. Thus the separation power of our spatial logic coincides with behavioral equivalence, even if it includes the basic structural connectives of composition and void, allowing it to *e.g.*, express arithmetical constraints on the number of sites in a system. We may however ask whether these structural operations are essential to characterize behavioral equivalence, in other words, whether the logic is minimal in some sense. We will give a positive answer to this question in the next section, in the more interesting case of weak behavioral equivalences.

3 Weak Bisimilarity

In this section we refine our previous results by considering a coarser observational equivalence, disregarding internal action, thus we adopt weak reduction barbed congruence as the reference observational equivalence. We denote by $\Rightarrow$ the reflexive-transitive closure of reduction ($\rightarrow$) and state that a network N weakly exhibits a barb a, noted $N \Downarrow_a$, if there is N' such that $N \Rightarrow N'$ and $N' \downarrow_a$. We then have:

Definition 3.1 [Weak reduction barbed congruence] *Weak reduction barbed congruence*, noted $\approxeq$, is the largest symmetric relation R such that for all $(N, M) \in R$:

For all barbs a, if $N\downarrow_a$ then $M\Downarrow_a$	(Barb closed)
If $N \to N'$ then there is M' s.t. $M \Rightarrow M'$ and $(N', M') \in R$	(Reduction closed)
For all contexts $C\,[\bullet]$, $(C\,[N]\,, C\,[M]) \in R$	(Context closed)

We establish some standard properties of weak reduction barbed congruence, such as $\approxeq$ is a congruence. We relate $\approxeq$ to the strong reduction barbed congruence.

Proposition 3.2 *We have* $\simeq \subset \approxeq$.

Proof. The proof of $\subseteq$ follows standard lines. To prove that $\simeq$ is strictly included in $\approxeq$ we may show that [**go**.**nil**] $\approxeq$ [**nil**] but [**go**.**nil**] $\not\simeq$ [**nil**]. □

Note that from Proposition 3.2 and Proposition 1.5 we immediately conclude $\equiv \subseteq \approxeq$. From the congruence property we obtain that reduction barbed congruence is closed under composition, which in particular for site composition gives us:

Lemma 3.3 *Let* P^i *and* Q^i $(i \in J)$ *be collections of processes. If for all* $i \in J$ *we have* $\big[P^i\big] \approxeq \big[Q^i\big]$, *then also* $\prod_{j\in J}\big[P^j\big] \approxeq \prod_{j\in J}\big[Q^j\big]$.

As for the strong case, weak reduction barbed congruence is already able to distinguish systems based on aspects of their structure, for instance, weak reduction barbed congruent networks always have the same number of sites. Also, as stated in Lemma 3.4, weak reduction barbed congruent networks weakly reduce to a one-one and onto correspondence of weakly reduction barbed congruent sites.

Lemma 3.4 *Let* M, N *be networks such that* $N \triangleq \prod_{j\in J}\big[P^j\big]$, *where* P^j $(j \in J)$ *is a collection of processes, and* $N \approxeq M$. *Then there is a collection of processes* Q^j $(j \in J)$ *such that* $M \Rightarrow \prod_{j\in J}\big[Q^j\big]$ *and for all* $j \in J$ *we have* $\big[P^j\big] \approxeq \big[Q^j\big]$.

Proof. (Sketch, full proof in [7]) The general idea is similar to that in the proof of Lemma 1.7. However, since now we may only weakly observe a barb, a different trick must be used to make sure that the migration of all the mark-placing processes has already occurred. We thus exploit the failure behavior of the context at a chosen point, avoiding in this way any chance for the migratory processes to postpone their choice of target, thus ensuring an unique correspondence. □

3.1 Weak Bisimilarity

We now propose a coinductive characterization of weak reduction barbed congruence. Weak commitment $\stackrel{\lambda}{\Longrightarrow}$ is the transition relation such that $N \stackrel{\lambda}{\Longrightarrow} N'$ when $N \stackrel{\tau}{\longrightarrow}^* M' \stackrel{\lambda}{\longrightarrow} M'' \stackrel{\tau}{\longrightarrow}^* N'$ and $\lambda \neq \tau$, and $N \stackrel{\tau}{\Longrightarrow} N'$ when $N \stackrel{\tau}{\longrightarrow}^* N'$. Given this we define weak bisimulations by adapting the labeled transition and separation clauses to the weak case. Notice that, contrasting with the strong case, the emptiness clause is essential here to distinguish, *e.g.*, [**nil**] from **0**.

Definition 3.5 [Weak Bisimulation] A binary relation $B \subseteq \mathcal{N} \times \mathcal{N}$ is a *weak bisimulation*

if and only if it is symmetric and whenever $(N, M) \in B$ then

$$N \equiv N' \mid N'' \Rightarrow \exists M', M'' \,.\, M \Rightarrow M' \mid M'' \wedge (N', M') \in B \wedge (N'', M'') \in B$$
$$N \equiv \mathbf{0} \Rightarrow M \equiv \mathbf{0}$$
$$N \xrightarrow{\lambda} N' \Rightarrow \exists M' \,.\, M \stackrel{\lambda}{\Longrightarrow} M' \wedge (N', M') \in B$$

We can prove that weak bisimulations enjoy usual properties, such as being equivalence relations, and closure under union. We thus define:

Definition 3.6 [Weak bisimilarity] *Weak bisimilarity*, noted $\approx$, is the largest weak bisimulation.

3.2 Full Abstraction

In this section, we prove that weak bisimilarity characterizes weak reduction barbed congruence in a fully abstract way, proof of which builds on the following results.

Lemma 3.7 *Let M be a network and P^j $(j \in J)$ a collection of processes such that $\prod_{j\in J}\left[P^j\right] \approx M$. Then there is a collection of processes Q^j $(j \in J)$ such that $M \Rightarrow \prod_{j\in J}\left[Q^j\right]$ and for all $j \in J$, $\left[P^j\right] \approx \left[Q^j\right]$.*

Proof. By induction on the size of J, using the separation and emptiness clauses. □

Lemma 3.8 is the cornerstone for proving full abstraction (Theorem 3.9). As for the strong case we must ensure compositionality at the process level due to process mobile capability, as process migration to sites results in inner site composition.

Lemma 3.8 *Let J be a finite set and I_j, for all $j \in J$, be a finite set. Let P_i^j and Q_i^j be processes such that for all $j \in J$ and $i \in I_j$ we have $\left[P_i^j\right] \approx \left[Q_i^j\right]$. Then*

$$\prod_{j\in J}\left[\prod_{i\in I_j} P_i^j\right] \approx \prod_{j\in J}\left[\prod_{i\in I_j} Q_i^j\right]$$

Proof. By coinduction on the definition of strong bisimulation. The proof follows the lines given for Lemma 2.5, with several adaptations needed for the weak case. Interesting to notice, in the strong case a migration of the inaction process could be mimicked by an internal computation, while here it can be mimicked by the empty sequence of internal actions (we no longer distinguish [**go**.**nil**] from [**nil**]). □

By Lemma 3.7 and Lemma 3.8 we prove that weak bisimilarity is a congruence, after which proof that $\approx \subseteq \approxeq$ follows in standard lines. To prove $\approxeq \subseteq \approx$ the difficulty lies in the spatial clauses, given by Lemma 3.3 and Lemma 3.4. Thus:

Theorem 3.9 (Full abstraction) *We have $\approx = \approxeq$.*

3.3 Logical Characterization of Weak Bisimilarity

We characterize weak bisimilarity (and thus weak reduction barbed congruence) using the spatial logic $\mathcal{L}_w$.

Definition 3.10 [Spatial Logic $\mathcal{L}_w$] Formulas are defined by the following syntax:

$$\text{(Formulas) } A, B, C ::= \mathbf{T} \;\Big|\; \neg A \;\Big|\; A \wedge B \;\Big|\; \mathbf{0} \;\Big|\; A \Uparrow B \;\Big|\; \langle\!\langle \lambda \rangle\!\rangle A$$

The logic $\mathcal{L}_w$ is obtained from $\mathcal{L}_s$ by adapting the composition operator, now noted $A \Uparrow B$, and the action modality, now noted $\langle\!\langle \lambda \rangle\!\rangle A$, to the weak case as defined in Definition 3.11. We leave the void operator with it's standard interpretation (notice that $N \Rightarrow \mathbf{0}$ is a trivial condition, due to the failure behavior).

Definition 3.11 [Semantics of $\mathcal{L}_w$] A formula's denotation is inductively given by

$$[\![\mathbf{T}]\!] \triangleq \mathcal{N} \quad [\![\neg A]\!] \triangleq \mathcal{N} \backslash [\![A]\!] \quad [\![A \wedge B]\!] \triangleq [\![A]\!] \cap [\![B]\!] \quad [\![\mathbf{0}]\!] \triangleq \{N \mid N \equiv \mathbf{0}\}$$
$$[\![A \Uparrow B]\!] \triangleq \{N \mid \exists N', N'' \,.\, N \Rightarrow N' \mid N'' \wedge N' \in [\![A]\!] \wedge N'' \in [\![B]\!]\}$$
$$[\![\langle\!\langle \lambda \rangle\!\rangle A]\!] \triangleq \{N \mid \exists N' \,.\, N \stackrel{\lambda}{\Longrightarrow} N' \wedge N' \in [\![A]\!]\}$$

We prove logical characterization of $\approx$, following the lines of Theorem 2.9.

Theorem 3.12 (Logical Characterization of $\approx$) *We have $\approx \;=\; =_{\mathcal{L}_w}$.*

As a corollary of Theorem 3.12 we conclude that the separation power of $\mathcal{L}_w$ precisely coincides with weak reduction barbed congruence, even if it includes the spatial operators composition and void. At this point, we may ask, as at the end of Section 2.2, whether the spatial operators are essential to the characterization. We may verify that $\mathbf{T}$ can be expressed as $\langle\!\langle \tau \rangle\!\rangle \mathbf{0}$, and $\langle\!\langle \tau \rangle\!\rangle A$ as $A \Uparrow \mathbf{0}$. Thus let $\mathcal{L}_w^{min}$ be the $(\mathbf{T}, \langle\!\langle \tau \rangle\!\rangle A)$-free fragment of $\mathcal{L}_w$. We may show that $\mathcal{L}_w^{min}$ is as expressive as $\mathcal{L}_w$, and moreover that all of its connectives are essential for its expressiveness.

Theorem 3.13 (Minimality) *The logic $\mathcal{L}_w^{min}$ is minimal. Moreover, the spatial operators are essential to characterize weak reduction barbed congruence.*

Proof. (Sketch, full proof in [7]) We show that any logic obtained from $\mathcal{L}_w^{min}$ by removing each connective is strictly less expressive.

- ($\neg A$) In the $\neg$-free fragment we are not able to express property $1 \triangleq \{N \mid \exists P \,.\, N \equiv [P]\}$, nor distinguish $[\mathbf{nil}] \mid [\mathbf{nil}]$ from $[\mathbf{nil}]$.
- ($A \wedge B$) In the $\wedge$-free fragment we can no longer express property 1.
- ($\mathbf{0}$) In the $\mathbf{0}$-free fragment we can no longer express property $\{N \mid N \equiv \mathbf{0}\}$, nor tell $\mathbf{0}$ and $[\mathbf{nil}]$ apart.
- ($A \Uparrow B$) In the $\Uparrow$-free fragment we can neither express property $2 \triangleq \{N \mid \exists P, Q \,.\, N \equiv [P] \mid [Q]\}$ nor separate $[\mathbf{nil}] \mid [\mathbf{nil}]$ from $[\mathbf{nil}]$.
- ($\langle\!\langle \alpha \rangle\!\rangle A$, $\alpha = \bar{a}, a$) The $\langle\!\langle \alpha \rangle\!\rangle$-free fragment does not tell $[\alpha.\mathbf{nil}]$ and $[\mathbf{nil}]$ apart.
- ($\langle\!\langle [a] \rangle\!\rangle A$) The $\langle\!\langle [a] \rangle\!\rangle$-free fragment does not distinguish $[\mathbf{go}.b.\mathbf{nil}]$ from $[\mathbf{nil}]$.

□

4 Concluding Remarks

We have studied observational equivalences in a distributed computation model, having obtained spatial logic characterizations of observational congruence in both the strong and weak cases. Our long term goal is to get a better understanding of how structural features contribute to observable distributed process behavior. Taking as reference semantics for observational congruence the standard reduction barbed congruence, we have derived equivalent characterizations of observational congruences in terms of co-inductively defined bisimilarities. The logics considered are natural extensions of HML with spatial operators, interpreted in the standard way.

We have thus shown, in a precise sense, that spatial logics, in particular the structural operators they offer, are not necessarily intensional, and may offer adequate expressive power for logically characterizing distributed behavior. We have also concluded, in the case of the specific process model here considered, that the composition operator $A \mid B$ is essential to capture (extensional) observational equivalence. Intuitively, such structural observations do not violate extensionality because distributed process behavior already has a related observational power, due to migration behavior and failures.

Observational equivalences of distributed systems have been studied extensively in the context of CCS-like models; a comprehensive survey may be found in [10]. However, it seems that logical characterizations have not been much discussed, and the distributed process equivalences proposed were technically defined by means of location or history-sensitive transition systems, where the use of location names plays a key role, both in the dynamic and static cases. Here, we build on a more abstract notion of spatial observation, avoiding the use of location names, and consider a calculus with anonymous sites, and migration primitives in the spirit of more recent proposals of calculi for distribution and mobility [8,19].

Our adoption of the simplest fail-stop failure model was motivated by the belief that it already captures the key consequences of failure, cf., the folklore slogan that in a distributed system one cannot distinguish a failed system from a system that will respond (much) later. The fail-stop model has been frequently adopted in formalizations of failure since [1], even if recent related works prefer to trigger failure by means of an explicit "kill" primitive [12]. Failures play an essential role in our results, even if, for the weak case, it is open whether failures, as we have modeled here, are absolutely essential. However, it is conceivable that other notions of failure, and a different set of spatial behaviors and spatial observations, may lead to results comparable to the ones reported in this paper.

It is interesting to compare our results with those of [14], where an extensional spatial logic (for the π-calculus) is considered. In that work, extensionality is obtained by removing the composition and void operators, while retaining the guarantee, whereas here we obtain extensionality by retaining the composition and void operators, while doing without the guarantee. We believe that the guarantee could be added to our developments, without breaking the results. Then, it would be instructive to see how to capture indirectly the action modalities, as in [15]. It would be certainly important to assess how to extend the general approach presented here to richer models, with name restriction, name passing, and full computational power.

Acknowledgments We acknowledge the Fundação para a Ciência e Tecnologia PhD Scholarship SFRH / BD / 23760 / 2005 and project IP Sensoria IST-2005-16004. We thank the anonymous reviewers for their comments, and Luís Monteiro and Luca Cardelli for useful remarks.

References

[1] Amadio, R. M. and S. Prasad, *Localities and Failures (Extended Abstract)*, in: P. S. Thiagarajan, editor, *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science **880** (1994), pp. 205–216.

[2] Caires, L., *Behavioral and spatial properties in a logic for the pi-calculus*, in: I. Walukiwicz, editor, *Proc. of Foundations of Software Science and Computation Structures'2004*, Lecture Notes in Computer Science (2004).

[3] Caires, L., *Proof Techniques for Distributed Resources and Behaviors using Spatial Logics*, in: *(discussion at) Symposium on Trustworthy Global Computing*, 2005.

[4] Caires, L. and L. Cardelli, *A Spatial Logic for Concurrency (Part I)*, Information and Computation **186** (2003), pp. 194–235.

[5] Caires, L. and E. Lozes, *Elimination of Quantifiers and Undecidability in Spatial Logics for Concurrency*, Theoretical Computer Science **10** (2006).

[6] Caires, L. and L. Monteiro, *Verifiable and Executable Specifications of Concurrent Objects in $\mathcal{L}_\pi$*, in: C. Hankin, editor, *7th European Symp. on Programming (ESOP 1998)*, number 1381 in Lecture Notes in Computer Science (1998), pp. 42–56.

[7] Caires, L. and H. T. Vieira, *Extensionality of Spatial Observations in Distributed Systems (Draft)*, Technical Report TR-DI/FCT/UNL-1/2006, DI/FCT Universidade Nova de Lisboa (2006), http://ctp.di.fct.unl.pt/~htv/pub/extspatial.pdf.

[8] Cardelli, L. and A. D. Gordon, *Mobile ambients*, in: M. Nivat, editor, *First International Conference on Foundations of Software Science and Computation Structures (FoSSaCS '98)*, Lecture Notes in Computer Science **1378** (1998).

[9] Cardelli, L. and A. D. Gordon, *Anytime, Anywhere. Modal Logics for Mobile Ambients*, in: *27th ACM Symp. on Principles of Programming Languages* (2000), pp. 365–377.

[10] Castellani, I., *Process algebras with localities*, in: J. Bergstra, A. Ponse and S. Smolka, editors, *Handbook of Process Algebra*, North-Holland, 2001 pp. 945–1045.

[11] Conforti, G., D. Macedonio and V. Sassone, *Spatial Logics for Bigraphs*, in: L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi and M. Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005*, Lecture Notes in Computer Science **3580** (2005), pp. 766–778.

[12] Francalanza, A. and M. Hennessy, *A Theory of System Behaviour in the Presence of Node and Link Failures*, in: M. Abadi and L. de Alfaro, editors, *CONCUR*, Lecture Notes in Computer Science **3653** (2005), pp. 368–382.

[13] Hennessy, M. and R. Milner, *Algebraic laws for Nondeterminism and Concurrency*, JACM **32** (1985), pp. 137–161.

[14] Hirschkoff, D., *An Extensional Spatial Logic for Mobile Processes*, in: P. Gardner and N. Yoshida, editors, *CONCUR 2004 15th International Conference*, Lecture Notes in Computer Science **3170** (2004), pp. 325–339.

[15] Hirschkoff, D., É. Lozes and D. Sangiorgi, *Minimality Results for the Spatial Logics*, in: P. K. Pandya and J. Radhakrishnan, editors, *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science **2914** (2003), pp. 252–264.

[16] Honda, K. and N. Yoshida, *On reduction-based process semantics*, Theoretical Computer Science **151** (1995), pp. 437–486.

[17] Milner, R. and D. Sangiorgi, *Barbed bisimulation*, in: W. Kuich, editor, *Automata, Languages and Programming, 19th International Colloquium*, Lecture Notes in Computer Science **623** (1992), pp. 685–695.

[18] Montanari, U. and V. Sassone, *Dynamic congruence vs. progressing bisimulation for ccs.*, Fundamenta Informaticae **16** (1992), pp. 171–199.

[19] Riely, J. and M. Hennessy, *Distributed processes and location failures*, Theor. Comput. Sci. **266** (2001), pp. 693–735.

[20] Sangiorgi, D., *Extensionality and Intensionality of the Ambient Logics*, in: *28th Annual Symposium on Principles of Programming Languages* (2001), pp. 4–13.

[21] Tuosto, E. and H. T. Vieira, *An observational model for spatial logics.*, Electronic Notes in Theoretical Computer Science **142** (2006), pp. 229–254.