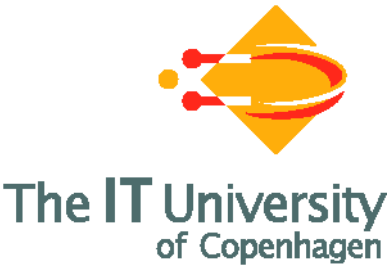


Preliminary Proceedings 15th International Workshop on Expressiveness in Concurrency

EXPRESS'08
Toronto, Canada
23 August 2008

Daniele Gorla (Università di Roma "La Sapienza")
Thomas Hildebrandt (IT University of Copenhagen)
editors

IT University Technical Report Series **TR-2008-108**

ISSN 1600–6100 **August 2008**

ISSN 1600–6100

ISBN 978-87-7949-181-6

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: +45 72 18 50 00
Telefax: +45 72 18 50 01
Web www.itu.dk

Contents

Preface

The EXPRESS workshops aim at bringing together researchers interested in the relations between various formal systems, particularly 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 till December 1997. The first three workshops were held respectively 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). EXPRESS'97, 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. EXPRESS'98, co-chaired by Ilaria Castellani and Catuscia Palamidessi, was held as a satellite workshop of the CONCUR'98 conference in Nice. Every year since then, EXPRESS has been a satellite workshop of the CONCUR conference. EXPRESS'99, co-chaired by Ilaria Castellani and Björn Victor, was hosted by the CONCUR'99 conference in Eindhoven. The EXPRESS'00 workshop, co-chaired by Luca Aceto and Björn Victor, was held as a satellite workshop of CONCUR 2000, Pennsylvania State University, USA. The EXPRESS'01 workshop, co-chaired by Luca Aceto and Prakash Panangaden, was held at BRICS, Aalborg University as a satellite of CONCUR'01. The EXPRESS'02 workshop, co-chaired by Uwe Nestmann and Prakash Panangaden, was held at Brno University as a satellite of CONCUR'02. The EXPRESS'03 workshop, co-chaired by Flavio Corradini and Uwe Nestmann, was co-located with CONCUR '03, Marseille, France. The EXPRESS '04 workshop, co-chaired by Jos Baeten and Flavio Corradini, was co-located with CONCUR '04, London, Great Britain. The EXPRESS '05 workshop, co-chaired by Jos Baeten and Iain Phillips, was co-located with CONCUR '05, San Francisco, USA. The EXPRESS '06 workshop, co-chaired by Roberto Amadio and Iain Phillips, was co-located with CONCUR '06, Bonn, Germany. The EXPRESS '07 workshop, co-chaired by Roberto Amadio and Thomas Hildebrandt, was co-located with CONCUR '07, Lissabon, Portugal.

This year EXPRESS is co-located with CONCUR '08, held in Toronto. In response to the call for papers, we received one short paper and 12 full papers. The program committee selected 8 of the full papers for presentation at the workshop. In addition, the workshop contains two invited presentations, by Michele Bugliesi (invited jointly with the SECCO workshop) and Gianluigi Zavattaro. The selected full papers and abstracts for the invited talks appear in these preliminary proceedings. We would like to thank the authors of the submitted papers, the invited speakers, the members of the program committee, and their subreferees for their contribution to both the meeting and this volume. Also we thank the CONCUR organising committee for hosting EXPRESS '08, and Richard Trefler for the local organization of the workshops and arranging the printing of these preliminary proceedings, which were compiled by Espen Højsgaard. The final proceedings will become available electronically at Elsevier's web site `http://www.elsevier.com/locate/entcs`.

The editors

Thomas Hildebrandt (IT University of Copenhagen)
Daniele Gorla (Università di Roma "La Sapienza")

EXPRESS 2008 Program Committee

Julian Bradfield, Edinburgh
Daniele Gorla (co-chair), Rome
Thomas Hildebrandt (co-chair), Copenhagen
Anna Ingólfsdóttir, Reykjavik
Alan Jeffrey, Bell-Labs, USA
Bas Luttik, Eindhoven
Sergio Maffeis, London
Gethin Norman, Oxford
Peter Selinger, Dalhousie
Frank Valencia, Paris
Daniele Varacca, Paris

EXPRESS 2008 Subreferees

Luca Aceto
Jesus Aranda
Silvio Capobianco
Marco Carbone
Pierre-Malo Denilou
Ton van Deursen
Murdoch Gabbay
Carlos Olarte
David Parker
Jorge Perez
Jon Sneyers
Martin Sulzmann
Emilio Tuosto

Security Abstractions and Adversarial Models in Distributed Communications (Abstract)

Michele Bugliesi [1] and Riccardo Focardi [2]

Dipartimento di Informatica
Università Ca' Foscari di Venezia

Security in distributed systems is notoriously hard to achieve due to the variety of tools and techniques available to an attacker interfering with/eavesdropping on/interrupting the communication links among the remote system components. The inherent difficulty in achieving the desired level of security creates a tension between two conflicting requirements in the design and specification of such system. One the one side, one must be very precise, and formal, in the specification of the security measures adopted to protect against the threats to which such systems are exposed. At the same time, however, one must be able to abstract away from such details and focus instead on the functional properties expected of the system.

In the literature on process calculi, this tension has generated a range of approaches to the specification of distributed system solutions, with two extremes: at one end, we find specification that draw on low-level cryptographic primitives as in the spi or applied-pi calculi, while on the other end lie specifications based on the pi-calculus, which assume very abstract, and hard-to-implement, mechanisms to secure communications by hiding them on private channels. A more recent line of research follows a different approach, aimed at identifying security primitives adequate as high-level programming abstractions, and at the same time well-suited for security analysis and verification in adversarial settings.

Following this more recent trend, we investigate a process calculus that that tries to strike a new balance between the formal simplicity deriving from high-level abstractions and the flexibility and expressive power required for the specification and implementation of realistic network applications. The calculus is based on security abstractions that support concise, high-level programming idioms for distributed security-sensitive applications, and at the same time are powerful enough to ex-

[1] Email: bugliesi@dsi.unive.it
[2] Email: focardi@dsi.unive.it

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

press a full-fledged adversarial setting. Drawing on this calculus, we will look at reasoning methods for security based on the long-established practice by which security properties are defined in terms of behavioral equivalences, and develop powerful up-to techniques to provide simple co-inductive proofs. We then contrast various adversarial models corresponding to different attacker's capabilities, compare their relative expressive power and study the relative strength of the corresponding behavioral equivalences for security.

This is a continuation of our initial work in [1].

References

[1] Bugliesi, M. and R. Focardi, *Language based secure communication*, in: *21st IEEE Computer Security Foundation Symposium* (2008), pp. 3–16.

EXPTIME-complete Decision Problems for Modal and Mixed Specifications[1]

Adam Antonik and Michael Huth[2,3]

Department of Computing, Imperial College London, United Kingdom

Kim G. Larsen and Ulrik Nyman[4,5]

Department of Computer Science, Aalborg University, Denmark

Andrzej Wąsowski[6]

IT University of Copenhagen, Denmark

Abstract

Modal and mixed transition systems are formalisms allowing mixing of over- and under-approximation in a single specification. We show EXPTIME-completeness of three fundamental decision problems for such specifications: whether a set of modal or mixed specifications has a common implementation, whether a sole mixed specification has an implementation, and whether all implementations of one mixed specification are implementations of another mixed or modal one. These results are obtained by a chain of reductions starting with the acceptance problem for linearly bounded alternating Turing machines.

1 Introduction

Behavioral models capture actual, desired or required system behavior and can so serve as documentation, specification or as the basis of analysis and validation activities. Formal behavioral models — of which we mention process algebras, Petri nets and labelled transition systems — bring a high degree of rigor and dependability to validation and verification activities.

[1] Huth and Antonik were partially supported by the UK EPSRC projects *Efficient Specification Pattern Library for Model Validation EP/D50595X/1* and *Complete and Efficient Checks for Branching-Time Abstractions EP/E028985/1*. Wąsowski was partially funded by CISS—Center for Embedded Software Systems, Aalborg University

[2] aa1001@doc.imperial.ac.uk

[3] mrh@doc.imperial.ac.uk

[4] kgl@cs.aau.dk

[5] ulrik@cs.aau.dk

[6] wasowski@itu.dk

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

Often one has to deal with more than one behavioral model at a time. For example, in requirement elaboration one may have several versions of a model, in component-based design one may have models that each focus on a different aspect of the system, and in formal verification one may have a system model accompanied by models that represent either desired features or genuinely faulty behavior. In each of these cases the modeller may want to have assurance that this collection of models is consistent. If versions of models are inconsistent with each other, this may reveal important implementation trade-offs. If all aspect models are inconsistent, their combination is not implementable. If a system model is inconsistent with all members of a given set of fault models, the system will not exhibit any of these flaws. Finally if a system model is consistent with a set of feature models, then the system will be able to actually implement all these features.

A related concept is the consistency of a *single* behavioral model. If models serve as specifications, their inconsistency suggests that the specification cannot be implemented. Conversely, a consistent model boosts our confidence of implementability and may even allow code-generation of such an implementation.

The stepwise-refinement paradigm proposes to write specifications as models and to then repeatedly refine such models until an implementation has been realized. In a *thorough* interpretation, refinement is decreasing the set of possible implementations: only implementations that were possible before the refinement step are still possible thereafter, but not necessarily all of them anymore.

This paper is devoted to studying the exact computational complexity of these three decision problems; whether finitely many models are consistent, whether a single model is consistent, and whether one model thoroughly refines another. The actual models we study are *mixed specifications* — stateful models with allowed and required transitions, well recognized as a formal foundation for system specification and abstraction alike [24,18,25,5,22,23,8,9,21,20]. We show that

- deciding whether finitely many modal or mixed specifications are consistent is EXPTIME-complete in the sum of the sizes of these specifications
- deciding whether one mixed specification is consistent is EXPTIME-complete in the size of that specification
- deciding whether one mixed specification thoroughly refines another mixed specification is EXPTIME-complete in the sum of their sizes.

Interestingly, checking the consistency of 100 mixed specifications with a few states each can be dramatically more complex than checking the consistency of a few mixed specifications with 100 states each. This is in striking contrast to the situation when all mixed specifications are fully refined (have identical required and allowed behaviors). In that case, consistency checks reduce to pairwise bisimilarity checks, which can be performed in polynomial time.

Our complexity results motivate future research that aims to either approximate these three decision problems soundly and efficiently, or that identifies sub-classes of specifications for which these decision problems are less complex.

We proceed by introducing the necessary background on alternating Turing machines, specifications, and their decision problems in Section 2. In Section 3 state-of-the-art bounds for these problems are reported. The new EXPTIME-completeness

results are given in Section 4. Section 5 reflects on a remaining open complexity gap for a special kind of mixed specifications, modal ones. We conclude in Section 6.

Related work

We refer to our recent overview [2] for a full account of related work. The present paper primarily improves on the results of [3], which are discussed in detail in Section 3. The relation of this work to generalized model checking [4] is detailed in Section 5.

In [13] a superpolynomial algorithm is given, which establishes common implementation for $k > 1$ modal specifications. The algorithm is exponential in k, but polynomial if k is fixed. It computes a common implementation if one exists. These upper bounds follow also from the polynomial algorithm for consistency checking of a conjunction of disjunctive modal transition systems, as studied in [25].

In [14] Hussain and Huth present an example of two modal specifications that have a common implementation but no greatest common implementation.

Fischbein et al. [10] use modal specifications for behavioral conformance checking of products against specifications of product families. They propose a new thorough refinement whose implementations are defined through a generalization of branching bisimulation. The thorough refinement obtained in this manner is finer than weak refinement, and argued to be more suitable for conformance checking.

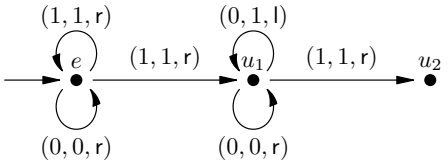
2 Background

Let us begin with a definition of the decision problem used in the main proof of this paper. An *Alternating Turing Machine* [6], or an ATM, is a tuple $T = (Q, \Gamma, \delta, q_0, \mathsf{mode})$, where Q is a non-empty finite set of control states, Γ is an alphabet of tape symbols, $\mathsf{null} \notin \Gamma$ is a special symbol denoting empty cell contents, $\delta\colon Q \times (\Gamma \cup \{\mathsf{null}\}) \to \mathcal{P}(Q \times \Gamma \times \{\mathsf{l}, \mathsf{r}\})$ is a transition relation, $q_0 \in Q$ is the initial control state, and $\mathsf{mode}\colon Q \to \{\mathsf{Univ}, \mathsf{Exst}\}$ is a labeling of control states as respectively universal or existential. Universal and existential states with no successors are called accepting and rejecting states (respectively). Each ATM T has an infinite tape of cells with a leftmost cell. Each cell can store one symbol from Γ. A head points to a single cell at a time, which can then be read or written to. The head can then move to the left or right: $(q', a', \mathsf{r}) \in \delta(q, a)$, e.g., says "if the head cell (say c) reads a at control state q, then a successor state can be q', in which case cell c now contains a' and the head is moved to the cell on the right of c." The state of the tape is an infinite word over $\Gamma \cup \{\mathsf{null}\}$.

Figure 1 presents an example of an ATM T over a binary alphabet $\Gamma = \{0, 1\}$ where arrows $q \xrightarrow{(a,a',d)} q'$ denote $(q', a', d) \in \delta(q, a)$. The initial control state e is an existential one, and both u_i control states are universal.

Configurations of an ATM T are triples $\langle q, i, \tau \rangle$ where $q \in Q$ is the current control state, the head is on the ith cell from the left, and $\tau \in (\Gamma \cup \mathsf{null})^\omega$ is the current tape state. For input $w \in \Gamma^*$, the initial configuration is $\langle q_0, 1, w\mathsf{null}^\omega \rangle$. The recursive and parallel execution of all applicable[7] transitions δ from initial configuration

[7] Transitions $(\_, \_, \_, \mathsf{l})$ are not applicable in configurations $\langle \_, 1, \_ \rangle$ as the head cannot move over the left



$$\delta(e,\ 0) = \{(e,\ 0, \mathsf{r})\}$$
$$\delta(e,\ 1) = \{(e,\ 1, \mathsf{r}), (u_1, 1, \mathsf{r})\}$$
$$\delta(u_1, 0) = \{(u_1, 1, \mathsf{l}), (u_1, 0, \mathsf{r})\}$$
$$\delta(u_1, 1) = \{(u_2, 1, \mathsf{r})\}$$
$$\delta(u_2, 0) = \delta(u_2, 1) = \{\}$$

Fig. 1. The transition relation of an ATM as a labelled graph and a function.

$\langle q_0, 1, w\mathsf{null}^\omega\rangle$ yields a computation tree $\mathsf{T}_{\langle T,w\rangle}$. We say that T accepts input w iff the tree $\mathsf{T}_{\langle T,w\rangle}$ accepts, where the latter is a recursive definition:

- $\mathsf{T}_{\langle T,w\rangle}$ with root $\langle q, i, \tau\rangle$ and $\mathsf{mode}(q) = \mathsf{Exst}$ accepts iff there is a successor $\langle q', i', \tau'\rangle$ of $\langle q, i, \tau\rangle$ in $\mathsf{T}_{\langle T,w\rangle}$ such that the sub-tree with root $\langle q', i', \tau'\rangle$ accepts
- $\mathsf{T}_{\langle T,w\rangle}$ with root $\langle q, i, \tau\rangle$ and $\mathsf{mode}(q) = \mathsf{Univ}$ accepts iff for all successors $\langle q', i', \tau'\rangle$ of $\langle q, i, \tau\rangle$ in $\mathsf{T}_{\langle T,w\rangle}$ the sub-tree with root $\langle q', i', \tau'\rangle$ accepts (in particular, this is the case if there are no such successors)

The ATM of Figure 1 accepts the regular language $(0+1)^*10^*1(0+1)^*$. Observe that u_2 is the only accepting state. Intuitively the part of T rooted in e accepts the prefix $(0+1)^*1$ — the semantics of existential states is locally that of states in non-deterministic Turing machines. The part of T rooted in u_1 consumes a series of 0 symbols until 1 is reached, which leads to acceptance. The suffix of the input word after the last 1 is ignored. Note that the computation forks in u_1 whenever a 0 is seen. However, the top branch would reach the earlier 1 eventually and accept.

An ATM T is *linearly bounded* iff for all words $w \in \Gamma^*$ accepted by T, the accepting part of the computation tree only contains configurations $\langle q, i, v\mathsf{null}^\omega\rangle$, where the length of $v \in \Gamma^*$ is no greater than the length of w. That is to say, by choosing exactly one accepting successor for each existential configuration in $\mathsf{T}_{\langle T,w\rangle}$, and by removing all the remaining successors and configurations unreachable from the root, one can create a smaller tree that only contains configurations with $\langle q, i, v\mathsf{null}^\omega\rangle$ where $|v| \leq |w|$. We refer to such pruned computation trees simply as "computations".

Our notion of "linear boundedness" follows [17] in limiting the tape size to the size of the input. This limitation does not change the hardness of the acceptance problem (see below). In addition we assume that linearly bounded ATMs have no infinite computations since any linearly bounded ATM can be transformed into another linearly bounded ATM, which accepts the same language, but also counts the number of computation steps used, rejecting any computation whose number of steps exceeds the number of possible configurations. [8]

Let $\mathrm{ATM}_{\mathrm{LB}} = \{\langle T, w\rangle \mid w \in \Gamma^*$ accepted by linearly bounded ATM $T\}$. The problem of deciding if for an arbitrary linearly bounded ATM T and an input w the pair $\langle T, w\rangle$ is in $\mathrm{ATM}_{\mathrm{LB}}$ is EXPTIME-complete [6].

Let us now define the basic models of interest in our study [18,8,7]:

Definition 2.1 For a finite alphabet of actions Σ, a *mixed specification* M is a

boundary of the tape.

[8] This is possible because ASPACE = EXPTIME [28, Thm. 10.18].

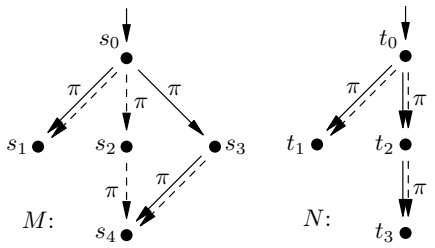


Fig. 2. Mixed $((M, s_0))$ and modal $((N, t_0))$ specifications with $I(M, s_0) = I(N, t_0)$ but not $(N, t_0) \prec (M, s_0)$.

triple $(S, R^{\Box}, R^{\Diamond})$, where S is a finite set of states and $R^{\Box}, R^{\Diamond} \subseteq S \times \Sigma \times S$ are the set of must- and may- transitions relations (respectively). A *modal* specification is a mixed specification satisfying $R^{\Box} \subseteq R^{\Diamond}$; all its must-transitions are also may-transitions. A *pointed* mixed (respectively modal) specification (M, s) is a mixed (modal) specification M with a designated initial state $s \in S$. The size $|M|$ of a mixed (modal) specification M is defined as $|S| + |R^{\Box} \cup R^{\Diamond}|$.

Refinement [18,8,7], called "modal refinement" in [21], is a co-inductive relationship between two mixed specifications that verifies that one such specification is more abstract than the other. This generalizes the co-inductive notion of bisimulation [26] to mixed specifications:

Definition 2.2 A mixed specification $(N, t_0) = ((S_N, R^{\Box}_N, R^{\Diamond}_N), t_0)$ *refines* another mixed specification $(M, s_0) = ((S_M, R^{\Box}_M, R^{\Diamond}_M), s_0)$ over the same alphabet Σ, written $(M, s_0) \prec (N, t_0)$, iff there is a relation $Q \subseteq S_M \times S_N$ containing (s_0, t_0) and whenever $(s, t) \in Q$ then

(i) for all $(s, a, s') \in R^{\Box}_M$ there exists some $(t, a, t') \in R^{\Box}_N$ with $(s', t') \in Q$
(ii) for all $(t, a, t') \in R^{\Diamond}_N$ there exists some $(s, a, s') \in R^{\Diamond}_M$ with $(s', t') \in Q$

Deciding whether one finite-state mixed specification refines another one is in P. For mixed specification (N, t_0) and modal specification (M, s_0) in Figure 2 we have $(M, s_0) \prec (N, t_0)$, given by $Q = \{(s_0, t_0), (s_1, t_1), (s_2, t_2), (s_3, t_2), (s_4, t_3)\}$. Note that throughout figures, solid arrows denote $R^{\Box}$-transitions, and dashed arrows denote $R^{\Diamond}$-transitions. But we do not have $(N, t_0) \prec (M, s_0)$. To see this, assume that there is a relation Q with $(t_0, s_0) \in Q$ satisfying the properties in Definition 2.2. Then from $(s_0, \pi, s_2) \in R^{\Diamond}_M$ we infer that there must be some x with $(t_0, \pi, x) \in R^{\Diamond}_N$ and $(x, s_2) \in Q$. In particular, x can only be t_1 or t_2. If x is t_1, then since $(s_2, \pi, s_4) \in R^{\Diamond}_M$ and $(t_1, s_2) \in Q$ there has to be some $R^{\Diamond}_N$ transition out of t_1, which is not the case. If x is t_2, then $(t_2, \pi, t_3) \in R^{\Box}_N$ and $(t_2, s_2) \in Q$ imply that there is some $R^{\Box}_M$ transition out of s_2, which is not the case. In conclusion, there cannot be such a Q and so $(N, t_0) \not\prec (M, s_0)$.

Labeled transition systems over an alphabet Σ are pairs (S, R) where S is a non-empty set of states and $R \subseteq S \times \Sigma \times S$ is a transition relation. We identify labelled transition systems (S, R) with modal specifications (S, R, R). The set of implementations $I(M, s)$ of a mixed specification (M, s) are all pointed labelled transition systems (T, t) refining (M, s). Note that $I(M, s)$ may be empty in general, but is guaranteed to be non-empty if M is a modal specification.

Definition 2.3 Let (N,t) and (M,s) be pointed mixed specifications. As in [21] we define *thorough refinement* $(M,s) \prec_{th} (N,t)$ to be the predicate $I(N,t) \subseteq I(M,s)$.

Refinement approximates this notion: $(M,s) \prec (N,t)$ implies $(M,s) \prec_{th} (N,t)$ since refinement is transitive. The converse is known to be false [16,29,27]; Figure 2 provides a counterexample.

We shall now formally define the decision problems informally stated above:

Common implementation (CI): given $k > 1$ modal or mixed specifications (M_i, s_i), is the set $\bigcap_{i=1}^{k} I(M_i, s_i)$ non-empty?

Consistency (C): Is $I(M,s)$ non-empty for a modal or mixed specification (M,s)?

Thorough refinement (TR): Does a mixed specification (N,t) thoroughly refine a mixed specification (M,s), i.e., do we have $I(N,t) \subseteq I(M,s)$?

As far as these decision problems are concerned, the restriction to finite implementations, which follows from restricting our definitions to finite specifications, causes no loss of generality, as already explained in [3]. A mixed specification (M,s) is consistent in the infinite sense iff its characteristic modal mu-calculus formula $\Psi_{(M,s)}$ [15] is satisfiable. Appealing to the small model theorem for mu-calculus, $\Psi_{(M,s)}$ is satisfiable iff it is satisfiable over finite-state implementations. We can reason in a similar manner about common implementation, which justifies the restriction to finite-state specifications and implementations.

Throughout this paper we work with Karp reductions, many-one reductions computable by deterministic Turing machines in polynomial time. This choice is justified since we reduce problems that are EXPTIME-complete.

3 Current Bounds

In [3], the three decision problems CI, C, and TR were studied for mixed and modal specifications. The results of [3] are summarized in Table 1. Two reductions were given in [3] that we appeal to here:

- a reduction of CI for modal specifications to C for *mixed* specifications
- a reduction of C for mixed specifications to TR for *mixed* specifications.

EXPTIME-hardness of CI for modal specifications would thus render EXPTIME-completeness of the decision problems CI, C, and TR for mixed specifications. We

Table 1
A summary given in [3] of the results provided in [3].

	Modal specifications	Mixed specifications
Common impl.	PSPACE-hard, EXPTIME	PSPACE-hard, EXPTIME
Consistency	trivial	PSPACE-hard, EXPTIME
Thorough ref.	PSPACE-hard, EXPTIME	PSPACE-hard, EXPTIME

turn to this EXPTIME-hardness proof in the next section.

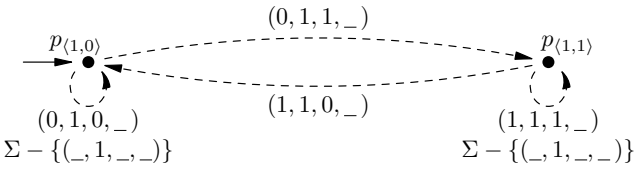


Fig. 3. Specification M_1 of the first tape cell in our running example, assuming $w_1 = 0$.

4 EXPTIME-Completeness Results

Theorem 4.1 *Let $\{(M_l, s_l)\}_{l\in\{1\ldots k\}}$ be a finite family of modal specifications over the same action alphabet Σ. Deciding whether there exists an implementation (I, i) such that $(M_l, s_l) \prec (I, i)$ for all $l = 1 \ldots k$ is EXPTIME-hard.*

We prove Theorem 4.1 by demonstrating a PTIME reduction from $\mathrm{ATM}_{\mathrm{LB}}$. Given an ATM T and an input word w of length n we synthesize a collection of (pointed) modal specifications $\mathcal{M}_w^T = \{M_i \mid 1 \leq i \leq n\} \cup \{M_{\mathrm{head}}, M_{\mathrm{ctrl}}, M_{\mathrm{exist}}\}$ whose sum of sizes is polynomial in n and in the size of T, such that T accepts w iff there exists an (pointed) implementation I refining all members of $\mathcal{M}_w^T$.

Specifications M_i, M_{head}, M_{ctrl}, and M_{exist} model tape cell i, the current head position, the finite control of T, and acceptance (respectively). Common implementations of these specifications model action synchronization to agree on what symbol is read from the tape, what is the head position, what is the symbol written to the tape, in what direction the head moves, and what are the transitions taken by the finite control, and whether a computation is accepting. The achieved effect is that a common refinement of these specifications corresponds to an accepting computation of T on input w. More precisely, any common implementations will correspond to different unfoldings of the structure of the finite control into a computation tree based on the content of the tape cells and the tape head position.

We now describe the specifications in $\mathcal{M}_w^T$ both formally and through our running example in Figure 1. All specifications in $\mathcal{M}_w^T$ have the same alphabet [9]

$$\Sigma = \{\pi, \exists\} \cup (\Gamma \times \{1..n\} \times \Gamma \times \{\mathsf{l}, \mathsf{r}\})$$

where $\exists$ and π are fresh symbols whose transitions encode logical constraints like disjunction and conjunction. All other actions are of the form (a_1, i, a_2, d) and denote that the machine's head is over the ith cell of the tape, which contains the a_1 symbol, and that it shall be moved one cell in the direction d after writing a_2 in the current cell. The alphabet for our running example is

$$\{\pi, \exists\} \cup (\{0,1\} \times \{1..n\} \times \{0,1\} \times \{\mathsf{l}, \mathsf{r}\}) \ .$$

Encoding Tape Cells.

For each tape cell i, specification M_i represents the possible contents of cell i. It has $|\Gamma|$ states $\{p_{\langle i,a\rangle}\}_{a\in\Gamma}$ and initial state $p_{\langle i,w_i\rangle}$, representing the initial contents

[9] A stricter and more complex reduction to CI of modal specifications over a *binary* alphabet is possible by encoding actions in binary form.

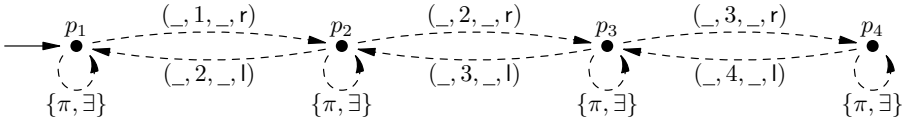


Fig. 4. Example of the head specification M_{head} assuming $|w| = 4$.

of the ith cell. There are no must-transitions:

$$R^{\Box} = \emptyset$$

The may-transition relation connects any two states:

$$\text{for all symbols } a_1,\ a_2 \text{ in } \Gamma \text{ we have } (p_{\langle i,a_1\rangle}, (a_1, i, a_2, \_), p_{\langle i,a_2\rangle}) \in R^{\diamond}$$

Changes in cells other than i are also consistent with M_i:

$$\text{for all } a \in \Gamma \text{ if } i \neq j,\ 1 \leq j \leq n, \text{ then } (p_{\langle i,a\rangle}, (\_, j, \_, \_), p_{\langle i,a\rangle}) \in R^{\diamond}$$

Finally the π and $\exists$ actions may be used freely as they do not affect the contents of the cell:

$$(p_{\langle i,a\rangle}, \pi, p_{\langle i,a\rangle}) \in R^{\diamond} \text{ and } (p_{\langle i,a\rangle}, \exists, p_{\langle i,a\rangle}) \in R^{\diamond} \text{ for any } a \in \Gamma$$

There are no more may-transitions in M_i.

Figure 3 presents a specification M_1 for the leftmost cell of an ATM over a binary alphabet. In figures we visualize multiple transitions with the same source and target as single arrows labeled with sets of actions. Several labels placed by the same arrow denote a union of sets. Wildcards (the '_' symbol) are used to generate sets of actions that match the pattern in the usual sense.

Encoding The Head.

Specification M_{head}, which tracks the current head position, has n states labeled p_1 to p_n — one for each possible position. Initially, the head occupies the leftmost cell, so p_1 is the initial state of M_{head}. There are no must-transitions:

$$R^{\Box} = \emptyset$$

May-transitions are consistent with any position changes based on the direction encoded in observed actions. More precisely,

$$\text{for every position } 1 \leq i < n \text{ we have } (p_i, (\_, i, \_, \mathsf{r}), p_{i+1}) \in R^{\diamond}$$

$$\text{for every } 1 < i \leq n \text{ we have } (p_i, (\_, i, \_, \mathsf{l}), p_{i-1}) \in R^{\diamond}$$

The π and $\exists$ transitions may again be taken freely, but in this case without moving the machine's head:

$$(p_i, \pi, p_i) \in R^{\diamond} \text{ and } (p_i, \exists, p_i) \in R^{\diamond} \text{ for each } 1 \leq i \leq n$$

There are no more may-transitions in M_{head}. Note that the head of T is only allowed to move between the first and nth cell in any computation. Figure 4 shows specification M_{head} for our running example.

Encoding The Finite Control.

Specifications M_{ctrl} and M_{exist} model the finite control of the ATM T. Specification M_{exist} is indepenendent of the ATM T. It is defined in Figure 5. It ensures that a π-transition is taken after every $\exists$-transition. Specification M_{ctrl} mimics the finite control of T almost directly. Each control state $q_s \in Q$ is identified with a state in M_{ctrl} of the same name. Additional internal states of M_{ctrl} encode existential and universal branching:

for each q_s a state $q_{s\exists}$ with two $\exists$-transitions $(q_s, \exists, q_{s\exists}) \in R^{\diamond} \cap R^{\Box}$ is added

Dependent on $\mathsf{mode}(q_s)$, additional states and transitions are created:

- If $\mathsf{mode}(q_s) = \mathsf{Exst}$: for each $1 \leq i \leq n$, $a_{\text{old}} \in \Gamma$, and for each transition $(q_t, a_{\text{new}}, d) \in \delta(q_s, a_{\text{old}})$ add a may π-transition from $q_{s\exists}$ to a new intermediate state uniquely named $\langle q_s a_{\text{old}} i a_{\text{new}} d q_t \rangle$, and add a must-transition labeled $(a_{\text{old}}, i, a_{\text{new}}, d)$ from that intermediate state to q_t. Formally:

$$(q_{s\exists}, \pi, \langle q_s a_{\text{old}} i a_{\text{new}} d q_t \rangle) \in R^{\diamond}$$
$$(\langle q_s a_{\text{old}} i a_{\text{new}} d q_t \rangle, (a_{\text{old}}, i, a_{\text{new}}, d), q_t) \in R^{\diamond} \cap R^{\Box} \ .$$

  Figure 6 shows this encoding for the state e of our running example.

- If $\mathsf{mode}(q_s) = \mathsf{Univ}$: for each $1 \leq i \leq n$, $a_{\text{old}} \in \Gamma$, and for each transition $(q_t, a_{\text{new}}, d) \in \delta(q_s, a_{\text{old}})$ add a may π-transition from $q_{s\exists}$ to an intermediate state named $\langle q_s a_{\text{old}} i \rangle$, and add a must-transition labeled $(a_{\text{old}}, i, a_{\text{new}}, d)$ from the intermediate state $\langle q_s a_{\text{old}} i \rangle$ to q_t. Formally:

$$(q_{s\exists}, \pi, \langle q_s a_{\text{old}} i \rangle) \in R^{\diamond} \ , \qquad (\langle q_s a_{\text{old}} i \rangle, (a_{\text{old}}, i, a_{\text{new}}, d), q_t) \in R^{\diamond} \cap R^{\Box} \ .$$

The initial state of M_{ctrl} is its state named q_0, where q_0 is the initial state of T. Figure 7 demonstrates the encoding of the state u_1 of the ATM in Figure 1. The complete specification M_{ctrl} for our running example is shown in Figure 8.

Notice how the two specifications M_{ctrl} and M_{exist} cooperate to enforce the nature of alternation. For example, for an existential state, M_{ctrl} forces every implementation to have an $\exists$-transition, which may be followed by a π-transition. Simultaneously M_{exist} allows an $\exists$-transition but requires a π-transition. Effectively at least one of the π branches from M_{ctrl} must be implemented (which is an encoding of a disjunction).

The complete family of modal specifications $\mathcal{M}_w^T$ contains all the specifications described above:

$$\mathcal{M}_w^T = \{M_i \mid 1 \leq i \leq n\} \cup \{M_{\text{head}}, M_{\text{ctrl}}, M_{\text{exist}}\}$$

Since the sum of their sizes is bounded by a polynomial in n and in the size of T, it remains to prove the following lemma:

Lemma 4.2 *For each linearly bounded ATM T and an input w, T accepts w iff the set of modal specifications $\mathcal{M}_w^T$ has a common implementation.*

The proof of Lemma 4.2 will appear in the final version of the paper. We mention here some points of interest. From an accepting computation tree $\mathsf{T}_{\langle T,w\rangle}$ one can construct a specification N by structural induction on $\mathsf{T}_{\langle T,w\rangle}$. This N effectively adds to $\mathsf{T}_{\langle T,w\rangle}$ some new states and labeled transitions so that the computation encoded in $\mathsf{T}_{\langle T,w\rangle}$ then interlocks with the action synchronization of specifications in $\mathcal{M}_w^T$. Since N is of the form (S, R, R) it suffices to show that N is a common refinement of all members in $\mathcal{M}_w^T$. This is a lengthy but routine argument.

For the converse, a common implementation of $\mathcal{M}_w^T$ is cycle-free by our assumption that T never repeats a configuration. So that pointed common implementation is a DAG and we use structural induction on that DAG to synthesize an accepting computation tree of T for input w. This makes use of the fact that the head of T never reaches a cell that was not initialized by input w.

Further results.

Theorem 4.1 states EXPTIME-hardness of CI for *modal* specifications. Together with the upperbound given in [3] we conclude that this bound is tight: CI is EXPTIME-complete. Moreover, by applying the reduction of CI for modal specifications to C for mixed specifications [3] we conclude that C for mixed specifications is EXPTIME-complete. Furthermore by appealing to the reduction of C for mixed specifications to TR for mixed specifications [3], we obtain that TR for mixed specifications is EXPTIME-complete as well.

Corollary 4.3 *The complexities shown in Table 2 are correct.*

5 Discussion

One complexity gap remains in Table 2, that for thorough refinement of *modal* specifications. Despite having made an extensive effort we can presently show neither EXPTIME-hardness nor membership in PSPACE for this problem.

In this context, it is useful to state that thorough refinement can be reduced to certain validity checks. First, as observed in [3], mixed and modal specifications (M, s) have characteristic formulæ $\Psi_{(M,s)}$ [15] in the modal μ-calculus such that pointed labeled transition systems (L, l) are implementations of (M, s) iff (L, l) satisfies $\Psi_{(M,s)}$. This was already observed in [19] for such formulæ written in vectorized form. So the thorough refinement problem of whether $(M, s) \prec_{th} (N, t)$ reduces to a validity check of $\neg\Psi_{(N,t)} \vee \Psi_{(M,s)}$. This raises the question of whether the validity problem for formulae given in the vectorized form of [19] is in PSPACE or whether it is EXPTIME-hard; that problem is known to be in EXPTIME (see

Table 2
Tabular summary of the results provided in this paper (in bold).

	Modal specifications	Mixed specifications
Common impl.	**EXPTIME-complete**	**EXPTIME-complete**
Consistency	trivial [24]	**EXPTIME-complete**
Thorough ref.	PSPACE-hard, EXPTIME [3]	**EXPTIME-complete**

for example [3]).

Second, we can reduce thorough refinement to a universal version of generalized model checking [4]. In loc. cit. Bruns and Godefroid consider judgments $\mathsf{GMC}(M, s, \varphi)$ which are true iff there exists an implementation of (M, s) satisfying φ. They remark that this generalizes both model checking (when (M, s) is an implementation) and satisfiability checking (when (M, s) is such that all labeled transition systems refine it). This existential judgment has a universal dual (see e.g. [1]), $\mathsf{VAL}(M, s, \varphi)$ which is true iff all implementations of (M, s) satisfy φ, thus generalizing model checking and validity checking. The former judgment is useful for finding counter-examples, the latter one for verification; e.g. both uses can be seen in the CEGAR technique for program verification of [11]. Since $(M, s) \prec_{th} (N, t)$ directly reduces to $\mathsf{VAL}(N, t, \Psi_{(M,s)})$, it would be of interest to understand the exact complexity of $\mathsf{VAL}(N, t, \varphi)$ for modal specifications (N, t) when φ ranges over characteristic formulæ $\Psi_{(M,s)}$ in vectorized form.

We remark that by translations and completeness results presented in [12] it follows that all complexity bounds presented here carry over to partial Kripke structures and Kripke modal transition systems.

6 Conclusion

We have discussed three fundamental decision problems for modal and mixed specifications: common implementation, consistency, and thorough refinement. For *modal* specifications, consistency is trivially true, while thorough refinement was previously shown to be PSPACE-hard and in EXPTIME [3]. For the remaining decision problems we have shown here that they are all EXPTIME-complete in the sum of the sizes of mixed or modal specifications.

We have appealed to known reductions between some of these problems [3] and, crucially, to a new reduction of input acceptance for linearly bounded alternating Turing machines to the existence of a common implementation for modal specifications – sketched in this extended abstract. The exact complexity of thorough refinement for modal specifications is subject to further investigation.

References

[1] Antonik, A. and M. Huth, *On the complexity of semantic self-minimization*, in: *Proc. AVOCS 2007*, to appear in ENTCS.

[2] Antonik, A., M. Huth, K. G. Larsen, U. Nyman and A. Wąsowski, *20 years of modal and mixed specifications*, Bulletin of EATCS (2008), available at http://processalgebra.blogspot.com/2008/05/concurrency-column-for-beatcs-june-2008.html.

[3] Antonik, A., M. Huth, K. G. Larsen, U. Nyman and A. Wąsowski, *Complexity of decision problems for mixed and modal specifications*, in: *FoSSaCS'08*, Lecture Notes in Computer Science **4962** (2008).

[4] Bruns, G. and P. Godefroid, *Generalized model checking: Reasoning about partial state spaces*, in: C. Palamidessi, editor, *CONCUR*, Lecture Notes in Computer Science **1877** (2000), pp. 168–182.

[5] Cerans, K., J. C. Godskesen and K. G. Larsen, *Timed modal specification - theory and tools*, in: *CAV '93: Proceedings of the 5th International Conference on Computer Aided Verification* (1993), pp. 253–267.

[6] Chandra, A. K., D. Kozen and L. J. Stockmeyer, *Alternation*, J. ACM **28** (1981), pp. 114–133.

[7] Clarke, E. M., O. Grumberg and D. E. Long, *Model checking and abstraction*, ACM Trans. Program. Lang. Syst. **16** (1994), pp. 1512–1542.

[8] Dams, D., "Abstract Interpretation and Partition Refinement for Model Checking," Ph.D. thesis, Eindhoven University of Technology (1996).

[9] Dams, D., R. Gerth and O. Grumberg, *Abstract interpretation of reactive systems*, ACM Trans. Program. Lang. Syst. **19** (1997), pp. 253–291.

[10] Fischbein, D., S. Uchitel and V. Braberman, *A foundation for behavioural conformance in software product line architectures*, in: *ROSATEA '06 Proceedings* (2006), pp. 39–48.

[11] Godefroid, P. and M. Huth, *Model checking vs. generalized model checking: Semantic minimizations for temporal logics*, in: *LICS* (2005), pp. 158–167.

[12] Godefroid, P. and R. Jagadeesan, *On the expressiveness of 3-valued models*, in: L. D. Zuck, P. C. Attie, A. Cortesi and S. Mukhopadhyay, editors, *VMCAI*, Lecture Notes in Computer Science **2575** (2003), pp. 206–222.

[13] Hussain, A. and M. Huth, *On model checking multiple hybrid views*, Technical report, Department of Computer Science, University of Cyprus (2004), TR-2004-6.
URL http://pubs.doc.ic.ac.uk/hybrid-logic-multiple-views/

[14] Hussain, A. and M. Huth, *Automata games for multiple-model checking*, Electr. Notes Theor. Comput. Sci. **155** (2006), pp. 401–421.

[15] Huth, M., *Labelled transition systems as a Stone space*, Logical Methods in Computer Science **1** (2005), pp. 1–28.
URL http://pubs.doc.ic.ac.uk/labelled-systems-metrics-Stone/

[16] Hüttel, H., "Operational and Denotational Properties of Modal Process Logic," Master's thesis, Computer Science Department. Aalborg University (1988).

[17] Laroussinie, F. and J. Sproston, *State explosion in almost-sure probabilistic reachability*, Inf. Process. Lett. **102** (2007), pp. 236–241.

[18] Larsen, K. G., *Modal specifications.*, in: J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science **407** (1989), pp. 232–246.

[19] Larsen, K. G., *Modal specifications.*, in: J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science **407** (1989), pp. 232–246.

[20] Larsen, K. G., U. Nyman and A. Wąsowski, *Modal I/O automata for interface and product line theories.*, in: R. D. Nicola, editor, *ESOP*, Lecture Notes in Computer Science **4421** (2007), pp. 64–79.

[21] Larsen, K. G., U. Nyman and A. Wąsowski, *On modal refinement and consistency*, in: L. Caires and V. T. Vasconcelos, editors, *CONCUR*, Lecture Notes in Computer Science **4703** (2007), pp. 105–119.

[22] Larsen, K. G., B. Steffen and C. Weise, *A constraint oriented proof methodology based on modal transition systems*, in: *Tools and Algorithms for Construction and Analysis of Systems*, 1995, pp. 17–40.
URL citeseer.ist.psu.edu/article/larsen95constraint.html

[23] Larsen, K. G., B. Steffen and C. Weise, *Fischer's protocol revisited: a simple proof using modal constraints*, Lecture Notes in Computer Science **1066** (1996), pp. 604–615.
URL citeseer.ist.psu.edu/larsen96fischers.html

[24] Larsen, K. G. and B. Thomsen, *A modal process logic*, in: *Third Annual IEEE Symposium on Logic in Computer Science (LICS)* (1988).

[25] Larsen, K. G. and L. Xinxin, *Equation solving using modal transition systems*, in: *Fifth Annual IEEE Symposium on Logics in Computer Science (LICS), 4–7 June 1990, Philadelphia, PA, USA*, 1990, pp. 108–117.

[26] Park, D., *Concurrency and automata on infinite sequences*, in: *Proceedings of the 5th GI-Conference on Theoretical Computer Science* (1981), pp. 167–183.

[27] Schmidt, H. and H. Fecher, *Comparing disjunctive modal transition systems with a one-selecting variant*, Submitted for publication (2007).
URL http://www.informatik.uni-kiel.de/~hf/papers/Fecher07CDMTSO-Sub.pdf

[28] Sipser, M., "Introduction to the Theory of Computation," International Thomson Publishing, 1996.

[29] Xinxin, L., "Specification and Decomposition in Concurrency," Ph.D. thesis, Department of Mathematics and Comnputer Science, Aalborg University (1992).

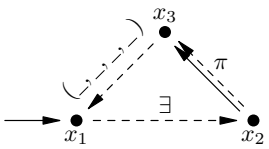


Fig. 5. Specification M_{exist} enforces a π-transition after each $\exists$-transition.

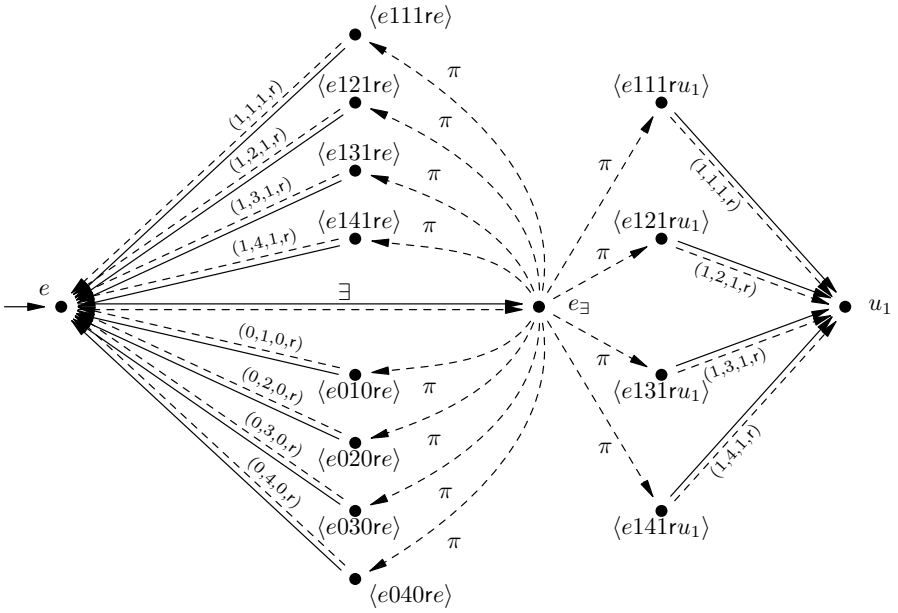


Fig. 6. Encoding for the existential state of the running example, assuming $|w| = 4$.

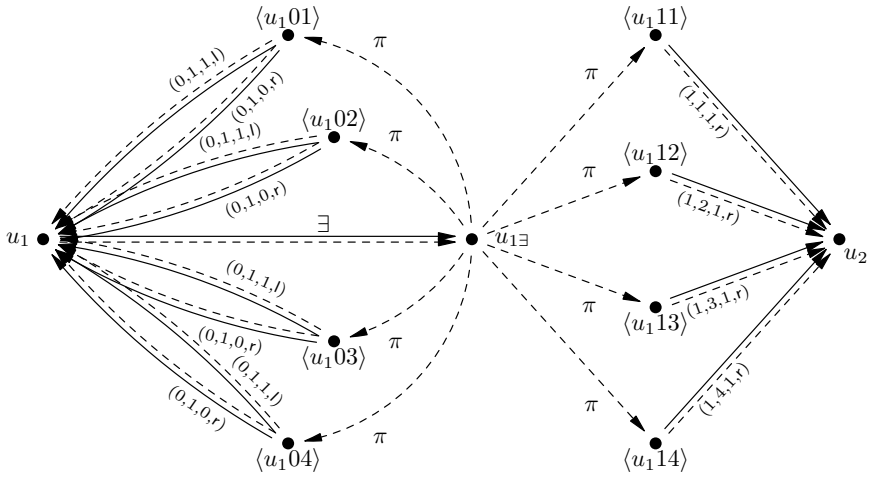


Fig. 7. Encoding for the universal state u_1 of the running example, assuming $|w| = 4$.

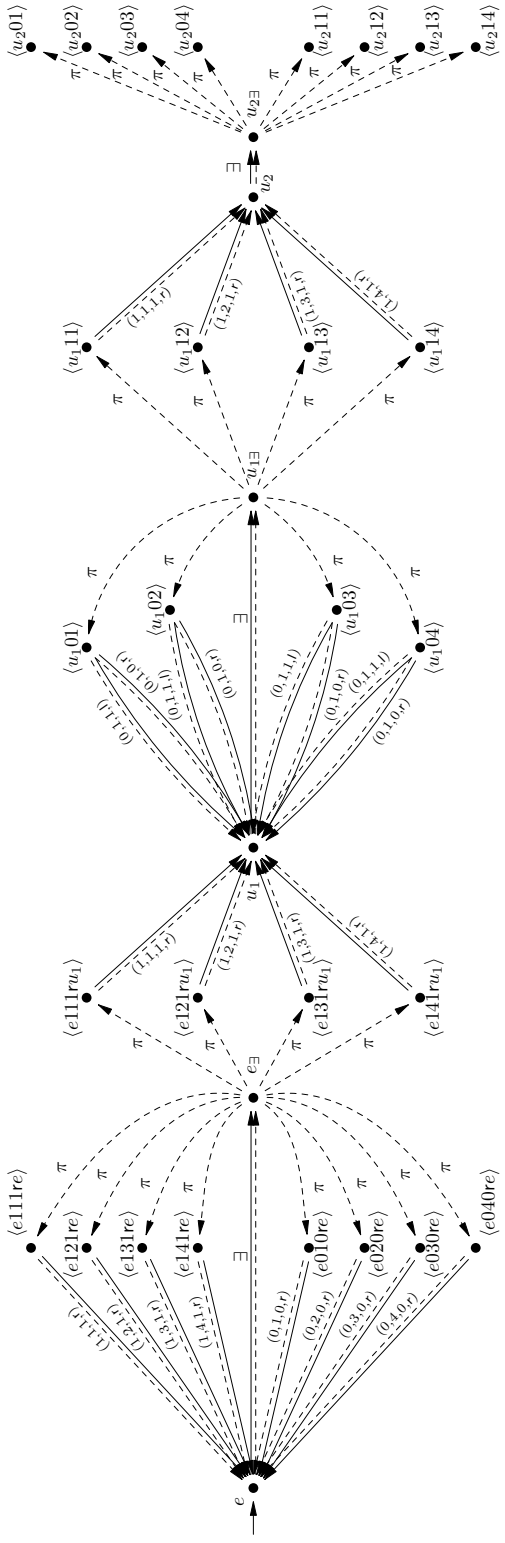


Fig. 8. The entire specification M_{ctrl} for the example of Figure 1 assuming $|w| = 4$.

A Basic Parallel Process as a Parallel Pushdown Automaton

J.C.M. Baeten[1] P.J.L. Cuijpers[1] P.J.A. van Tilburg[1]

Division of Computer Science, Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

We investigate the set of basic parallel processes, recursively defined by action prefix, interleaving, **0** and **1**. Different from literature, we use the constants **0** and **1** standing for unsuccessful and successful termination in order to stay closer to the analogies in automata theory.
We prove that any basic parallel process is rooted branching bisimulation equivalent to a regular process communicating with a bag (also called a parallel pushdown automaton) and therefore we can regard the bag as the prototypical basic parallel process.
This result is closely related to the fact that any context-free process is either rooted branching bisimulation equivalent or contrasimulation equivalent to a regular process communicating with a stack, a result that is the analogy in process theory of the language theory result that any context-free language is the language of a pushdown automaton.

Keywords: automata theory, process algebra, basic parallel process, parallel pushdown automaton

1 Introduction

In this paper, we study the class of basic parallel processes. This class was introduced in [7] as the class of all processes that have a finite guarded recursive specification over the small process algebraic language with **0**, action prefix, choice and parallel composition without communication (just interleaving). More work about this class can be found in e.g. [8,11]. Some results correspond to analogous results in formal language theory, such as the fact that every basic parallel language can be presented as a parallel pushdown automaton (a pushdown automaton not with a stack but with a *bag*, a multiset of variables).

However, there is an important difference between automata theory on the one hand and process algebra (CCS style) on the other hand that has been mostly neglected so far. In an automaton, for instance a non-deterministic finite automaton, any subset of the set of states can be marked as final, and for the definition of the language of an automaton only sequences that lead from the initial state to a final

[1] Email: {j.c.m.baeten,p.j.l.cuijpers,p.j.a.v.tilburg}@tue.nl.

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

state count. In this sense, successful termination in an automaton is observable. In process algebra CCS style, the only observables are executions of actions, with $\mathbf{0}$ being the process characterized by allowing no actions at all. Sequential composition can be defined, nevertheless, by having special 'tick' actions that by synchronization turn into internal actions. In process algebra ACP style, observables are action executions and action executions leading to termination. Sequential composition then becomes a basic operator. In both the CCS and ACP approaches, however, termination occurring in a choice context (a terminating state with an outgoing edge) cannot be presented accurately. This can be achieved with the introduction of the $\mathbf{1}$ process (characterizing a process that can only terminate), resulting in a full analogy with automata theory. Using this analogy, we can say that a regular process is the bisimulation equivalence class of a non-deterministic finite automaton, and the set of regular processes is exactly the set of processes given by a finite guarded recursive specification over $\mathbf{0}, \mathbf{1}$, action prefix and choice.

We investigated the set of context-free processes (defined with $\mathbf{1}$) in [4]. There, the addition of $\mathbf{1}$ makes an essential difference: a process can be defined that has unbounded branching, something that cannot be done without $\mathbf{1}$. Furthermore, we established in [4] under what conditions a context-free process can be presented as a pushdown automaton. In this paper, we investigate a similar result for the class of basic parallel processes. For basic parallel processes, the added expressivity is less spectacular (a corollary of our main theorem is that basic parallel processes have bounded branching, even those including $\mathbf{1}$), but still, without $\mathbf{1}$ a bag process expressed as a basic parallel process cannot be tested for being empty. In general, adding $\mathbf{1}$ makes that the theory becomes more challenging, and in our opinion also more interesting.

Another difference between automata theory and process theory is that process theory allows us to make communication explicit and abstract from it modulo branching bisimulation. In a setting with explicit communication, a pushdown automaton can be seen as a regular process communicating with a stack. Since every context-free process can be realized in this way, and the stack is a context-free process itself, we can look upon the stack as the prototypical context-free process. Similarly, we show in this paper that every basic parallel process can be presented as a regular process communicating with a bag, a multiset of data elements. Since the bag is a basic parallel process itself, it can be seen as the prototypical basic parallel process.

Thus, the result of [11] that every basic parallel process can be given by means of a parallel pushdown automaton is given here in an extended setting, with the process $\mathbf{1}$ and with explicit communication.

2 Regular Processes

Before we introduce the basic parallel processes, we first consider the notion of a regular process and its relation to regular languages in automata theory. We start with the definition of the notion of transition system from process theory. A finite transition system can be thought of as a non-deterministic finite automaton. In order to have a complete analogy, the transition systems we study have a subset of

states marked as final states.

Definition 2.1 (Transition system) *A* transition system *M is a quintuple $(\mathcal{S}, \mathcal{A}, \rightarrow, \uparrow, \downarrow)$ where:*

(i) *$\mathcal{S}$ is a set of states,*

(ii) *$\mathcal{A}$ is an alphabet,*

(iii) *$\rightarrow \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the set of* transitions *or* steps*,*

(iv) *$\uparrow \in \mathcal{S}$ is the initial state,*

(v) *$\downarrow \subseteq \mathcal{S}$ is a set of final states.*

For $(s, a, t) \in \rightarrow$ we write $s \xrightarrow{a} t$. For $s \in \downarrow$ we write $s\downarrow$. A finite transition system *or* non-deterministic finite automaton *is a transition system of which the sets $\mathcal{S}$ and $\mathcal{A}$ are finite.*

In accordance with automata theory, where a *regular language* is a language equivalence class of a non-deterministic finite automaton, we define a *regular process* to be a bisimulation equivalence class of a finite transition system. Contrary to automata theory, it is well-known that not every regular process has a *deterministic* finite transition system (i.e. a transition system for which the relation $\rightarrow$ is functional). The set of deterministic regular processes is a proper subset of the set of regular processes.

Next, consider the automata theoretic characterization of a regular language by means of a right-linear grammar. In process theory, a grammar is called a *recursive specification*: it is a set of recursive equations over a set of variables. A right-linear grammar then coincides with a recursive specification over a finite set of variables in the Minimal Algebra MA. (We use standard process algebra notation as propagated by [2,3].)

Definition 2.2 *The signature of* Minimal Algebra *MA is as follows:*

(i) *There is a constant* **0***; this denotes inaction, a deadlock state; other names are δ or stop.*

(ii) *There is a constant* **1***; this denotes termination, a final state; other names are ε, skip or the empty process.*

(iii) *For each element of the alphabet $\mathcal{A}$ there is a unary operator $a\_$ called* action prefix*; a term $a.x$ will execute the elementary action a and then proceed as x.*

(iv) *There is a binary operator $+$ called* alternative composition*; a term $x + y$ will either execute x or execute y, a choice will be made between the alternatives.*

The constants **0** and **1** are needed to denote transition systems with a single state and no transitions. The constant **0** denotes a single state that is not a final state, while **1** denotes a single state that is also a final state.

Definition 2.3 *Let $\mathcal{V}$ be a set of* variables*. A* recursive specification *over $\mathcal{V}$ with initial variable $S \in \mathcal{V}$ is a set of equations of the form $X = t_X$, exactly one for each $X \in \mathcal{V}$, where each right-hand side t_X is a term over some signature, possibly containing elements of $\mathcal{V}$. A recursive specification is called finite, if $\mathcal{V}$ is finite.*

We find that a finite recursive specification over MA can be seen as a right-linear grammar. Now each finite transition system corresponds directly to a finite recursive specification over MA, using a variable for every state. To go from a term over MA to a transition system, we use *structural operational semantics* [1], with rules given in Table 1.

$$\frac{}{\mathbf{1}\downarrow} \qquad \frac{}{a.x \xrightarrow{a} x}$$

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \qquad \frac{x\downarrow}{x + y\downarrow} \qquad \frac{y\downarrow}{x + y\downarrow}$$

$$\frac{t_X \xrightarrow{a} x \quad X = t_X}{X \xrightarrow{a} x} \qquad \frac{t_X\downarrow \quad X = t_X}{X\downarrow}$$

Table 1
Operational rules for MA and recursion ($a \in \mathcal{A}, X \in \mathcal{V}$).

3 Basic Parallel Processes

The class of basic parallel processes introduced by Christensen in [7] contains processes that can interleave actions of parallel components. In [4], we established that context-free processes can be given by recursive specifications over the Sequential Algebra SA, which extends MA with the *sequential composition* operator $\_ \cdot \_$. In this paper, we give parallel processes, a superset of the basic parallel processes, by recursive specifications over the *Communication Algebra* CA, which extends MA with the *parallel composition* operator $\_ \parallel \_$.

Now, consider the notion of a parallel pushdown automaton. A parallel pushdown automaton is a finite automaton, but at every step it can insert a number of elements into a bag by communicating along port i, or it can remove a single item from the bag by communicating along port o, and take this information into account in determining its next move. Thus, making the interaction explicit, a parallel pushdown automaton is a regular process communicating with a bag. In order to model the interaction between the regular process and the bag, again we use the Communication Algebra CA. We use a particular communication function, that will only synchronize actions $!_c d$ and $?_c d$ (for the same channel $c \in \{i, o\}$ and data element $d \in D$). The result of such a synchronization is denoted $⁉_c d$. Furthermore, CA contains the *encapsulation operator* $\partial_*(\_)$, which blocks actions $!_i d$, $?_i d$, $!_o d$ and $?_o d$, and the *abstraction operator* $\tau_*(\_)$ which turns all $⁉_i d$ and $⁉_o d$ actions into the internal action τ. For CA, we extend the operational rules of MA (see Table 1) with operational rules in Table 2.

Consider the following specification:

$$P = \mathbf{1} + P \parallel a.\mathbf{1}.$$

Our first observation is that, by means of the operational rules, we derive an infi-

$$\frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} \qquad \frac{x\downarrow \quad y\downarrow}{x \parallel y\downarrow}$$

$$\frac{x \xrightarrow{?_c d} x' \quad y \xrightarrow{!_c d} y'}{x \parallel y \xrightarrow{⁉_c d} x' \parallel y'} \qquad \frac{x \xrightarrow{!_c d} x' \quad y \xrightarrow{?_c d} y'}{x \parallel y \xrightarrow{⁉_c d} x' \parallel y'}$$

$$\frac{x \xrightarrow{a} x' \quad a \notin \{!_c d, ?_c d\}}{\partial_*(x) \xrightarrow{a} \partial_*(x')} \qquad \frac{x\downarrow}{\partial_*(x)\downarrow}$$

$$\frac{x \xrightarrow{⁉_c d} x'}{\tau_*(x) \xrightarrow{\tau} \tau_*(x')} \qquad \frac{x \xrightarrow{a} x' \quad a \neq ⁉_c d}{\tau_*(x) \xrightarrow{a} \tau_*(x')} \qquad \frac{x\downarrow}{\tau_*(x)\downarrow}$$

Table 2
Operational rules for CA ($a \in \mathcal{A}$, $c \in \{i, o\}$).

nite transition system, which moreover is infinitely branching. All the states of this transition system are different in bisimulation semantics, and so this is in fact an infinitely branching process. Our second observation is that this recursive specification has infinitely many different (non-bisimilar) solutions in the transition system model. This is because the equation is *unguarded*, the right-hand side contains a variable that is not in the scope of an action-prefix operator, and also cannot be brought into such a form. So, if there are multiple solutions to a recursive specification, we have multiple processes that correspond to this specification. This causes additional difficulties.

These two observations are the reason to restrict to guarded recursive specifications only. It is well-known that a guarded recursive specification has a unique solution in the transition system model (see [6,5]), and we show later on that this solution is also finitely branching. This restriction leads to our following definition of the basic parallel processes, a subclass of the parallel processes given by recursive specifications over CA.

Definition 3.1 *A* basic parallel process *is the bisimulation equivalence class of the transition system generated by a finite guarded recursive specification over the Communication Algebra CA such that the process only interleaves actions and synchronizes termination, but does not allow for communication to happen. That is, only the operational rules on the top line in Table 2 are used.*

In this paper, we use equational reasoning to manipulate recursive specifications. Our finite axiomatization of transition systems of CA modulo *rooted branching bisimulation* [9] uses the auxiliary operators $\_ \between \_$ and $\_ \mid \_$ [6,10]. See Table 3. See [3] for an explanation of the axioms.

Besides these axioms we use the Cluster Fair Abstraction Rule for (rooted) branching bisimulation CFAR^b, introduced in [5,12]. For a guarded recursive specification E and the set of abstractions $I \subseteq \mathcal{A}$ we want to abstract from, a subset C of the variables $\mathcal{V}$ is called *a cluster of I in E* if for all $X \in C$, the equation of X

$x \parallel y$	$= x \between y + y \between x + x \mid y$		$a.(\tau.(x+y)+x)$	$= a.(x+y)$	
$\mathbf{0} \between x$	$= \mathbf{0}$		$x \mid y$	$= y \mid x$	
$\mathbf{1} \between x$	$= \mathbf{0}$		$x \parallel \mathbf{1}$	$= x$	
$a.x \between y$	$= a.(x \parallel y)$		$\mathbf{1} \mid x + \mathbf{1}$	$= \mathbf{1}$	
$(x+y) \between z$	$= x \between z + y \between z$		$(x \parallel y) \parallel z$	$= x \parallel (y \parallel z)$	
$\mathbf{0} \mid x$	$= \mathbf{0}$		$(x \mid y) \mid z$	$= x \mid (y \mid z)$	
$(x+y) \mid z$	$= x \mid z + y \mid z$		$(x \between y) \between z$	$= x \between (y \parallel z)$	
$\mathbf{1} \mid \mathbf{1}$	$= \mathbf{1}$		$(x \mid y) \between z$	$= x \mid (y \between z)$	
$a.x \mid \mathbf{1}$	$= \mathbf{0}$		$x \between \tau.y$	$= x \between y$	
$!_c d.x \mid ?_c d.y$	$= ?_c d.(x \parallel y)$		$x \mid \tau.y$	$= \mathbf{0}$	
$a.x \mid b.y$	$= \mathbf{0}$	*if* $\{a,b\} \neq \{!_c d, ?_c d\}$			
$\partial_*(\mathbf{0})$	$= \mathbf{0}$		$\tau_*(\mathbf{0})$	$= \mathbf{0}$	
$\partial_*(\mathbf{1})$	$= \mathbf{1}$		$\tau_*(\mathbf{1})$	$= \mathbf{1}$	
$\partial_*(!_c d.x)$	$= \partial_*(?_c d.x) = \mathbf{0}$		$\tau_*(?_c d.x)$	$= \tau.\tau_*(x)$	
$\partial_*(a.x)$	$= a.\partial_*(x)$	*if* $a \notin \{!_c d, ?_c d\}$	$\tau_*(a.x)$	$= a.\tau_*(x)$	*if* $a \neq ?_c d$
$\partial_*(x+y)$	$= \partial_*(x) + \partial_*(y)$		$\tau_*(x+y)$	$= \tau_*(x) + \tau_*(y)$	

Table 3
Equational theory of CA ($a \in \mathcal{A} \cup \{\tau\}$, $c \in \{i, o\}$).

in E is of the form

$$X = \sum_{1 \leq k \leq m} i_k.X_k + \sum_{1 \leq j \leq n} Y_j,$$

where $i_1, \ldots, i_m \in I \cup \{\tau\}$, $X_1, \ldots, X_m \in C$, and $Y_1, \ldots, Y_n \in \mathcal{V} - C$. We call the set of variables $\{Y_1, \ldots, Y_n\}$ the *exits* of X, denoted with $U(X)$, and use $U(C)$ to refer to the exit set of the cluster C. The cluster C is called *conservative* if every exit from $U(C)$ is reachable from every variable in the cluster by doing a number of steps from $I \cup \{\tau\}$. Now, CFAR^b is the following rule:

$$\frac{E \text{ guarded} \qquad X \in C \qquad I \subseteq \mathcal{A} \qquad C \text{ is a finite conservative cluster of } I \text{ in } E}{\tau.\tau_I(X) = \tau. \sum_{Y \in U(C)} \tau_I(Y)}.$$

Furthermore, we often use the aforementioned principle that guarded recursive specifications have unique solutions [5].

The given equational theory is sound and ground-complete for the model of transition systems modulo rooted branching bisimulation [9,3]. This is the preferred model we use, but all our reasoning in the following takes place in the equational theory, so is model-independent provided the models preserve validity of the axioms, unique solutions for guarded recursive specifications and CFAR^b.

Using the axioms, any guarded recursive specification can be brought into *Greibach normal form* [7]:

$$X = \sum_{i \in I_X} a_i.\xi_i \ (+ \mathbf{1}).$$

In this form, every right-hand side of every equation consists of a number of summands, indexed by a finite set I_X (the empty sum is $\mathbf{0}$), each of which is $\mathbf{1}$, or of the form $a_i.\xi_i$, where ξ_i is the parallel composition of a number of variables (the empty multiset is $\mathbf{1}$). For a recursive specification in Greibach normal form, every state of the transition system is given by a multiset of variables just like in [11].

Note that we can take the index sets associated with the variables to be disjoint. As an example, we consider the important basic parallel process *bag*. Suppose D is a finite data set, then we define the following actions in $\mathcal{A}$, for each $d \in D$:

- $?_i d$: insert (push) d into the bag over the input channel i;
- $!_o d$: remove d from the bag over the output channel o.

Now the recursive specification is as follows:

$$B = \mathbf{1} + \sum_{d \in D} ?_i d.(B \parallel !_o d.\mathbf{1}).$$

In order to see that the above process indeed defines a bag, define processes B_μ, denoting the bag with contents $\mu \in D^*$, as follows: the first equation for the empty bag, the second for any nonempty bag, with isolated element d and rest bag μ (denoted with $\{d\} \uplus \mu$, but abbreviated with the notation $d\mu$ from here on):

$$\begin{aligned} B_\emptyset &= B, \\ B_{d\mu} &= !_o d.\mathbf{1} \parallel B_\mu. \end{aligned}$$

We obtain the following specification for the bag in Greibach normal form:

$$\begin{aligned} B_\emptyset &= \mathbf{1} + \sum_{d \in D} ?_i d.B_d, \\ B_{d\mu} &= !_o d.B_\mu + \sum_{e \in D} ?_i e.B_{ed\mu}. \end{aligned}$$

Finally, we define the *forgetful bag*, which can terminate even if it is not empty, as follows:

$$B = \mathbf{1} + \sum_{d \in D} ?_i d.\big(B \parallel (!_o d.\mathbf{1} + \mathbf{1})\big).$$

Note that while the bag is given by a recursive specification of CA, it is a basic parallel process, since no communication is possible between $!_o d$ and $?_i d$ for any $d \in D$.

4 Parallel Pushdown Automata

The main goal of this paper is to prove that every basic parallel process is equal to a regular process communicating with a bag. Thus, if P is any basic parallel process, then we want to find a regular process Q such that

$$P = \tau_*(\partial_*(Q \parallel B)),$$

where B is a (partially) forgetful bag process specified below.

Without loss of generality, we assume in this section that P is given in Greibach normal form. The data set D we use for our solution is the set of variables $\mathcal{V}$ of P. We call a variable *transparent* if its equation has a $\mathbf{1}$-summand. We denote the set of transparent variables of P with $\mathcal{V}^{+\mathbf{1}}$. Furthermore, we define the conditional process $\mathbf{1}_\xi$ as $\mathbf{1}$ if all variables in set or multiset ξ are transparent and as $\mathbf{0}$ otherwise.

Now, we prove the main theorem by first stating the specification of our solution, then proving necessary lemmas related to this specification before finally giving the main proof.

Theorem 4.1 *For every basic parallel process P there exists a process Q given by a finite guarded recursive specification over MA such that $P = \tau_*(\partial_*(Q \parallel B_\emptyset)) = [Q \parallel B_\emptyset]_*$* [2] *where B is the (partially) forgetful bag.*

Proof Let E be a finite recursive specification of P in Greibach normal form. Now, let F be a recursive specification that defines a parallel pushdown automaton. This specification contains the following equations for every variable $X \in \mathcal{V}$ of the specification E:

$$\hat{X} = \sum_{i \in I_X} a_i.\mathrm{Push}(\xi_i) + \mathbf{1}_X,$$

with $\mathrm{Push}(\xi)$ recursively defined as

$$\begin{aligned} \mathrm{Push}(\emptyset) &= \mathrm{Ctrl}, \\ \mathrm{Push}(X\xi') &= !_i X.\mathrm{Push}(\xi). \end{aligned}$$

where X is a variable that is in the original multiset ξ and ξ' is the multiset that is left over when X has been removed.

Additionally, let F contain the following equations of a *partially forgetful bag* and a (regular) finite control:

$$\begin{aligned} B &= \mathbf{1} + \sum_{\substack{V \in \mathcal{V} \\ V \notin \mathcal{V}^{+\mathbf{1}}}} ?_i V.(!_o V.\mathbf{1} \parallel B) + \sum_{\substack{V \in \mathcal{V} \\ V \in \mathcal{V}^{+\mathbf{1}}}} ?_i V.(!_o V.\mathbf{1} + \mathbf{1} \parallel B), \\ \mathrm{Ctrl} &= \sum_{V \in \mathcal{V}} ?_o V.(\hat{V} + !_i V.\mathrm{Ctrl}). \end{aligned}$$

The specification of each $\hat{X}$ in F mimics the behavior of each X in E by performing the same actions a_i and subsequently inserting each variable of the parallel composition ξ_i in the (partially) forgetful bag B. Once this is done, the process Ctrl arbitrarily removes a variable V from the bag (which is the multiset of variables that can be executed in parallel at this moment) and executes $\hat{V}$. Note that Ctrl doesn't make a choice because it can choose to reinsert the variable V and remove another one.

We interpret the multisets in Greibach normal forms as parallel compositions. In Greibach normal form, every state in P is labeled with a parallel composition of variables ξ. Substituting the Greibach normal form for the variables $X_1, \ldots, X_n$

[2] From here on, $[p]_*$ is used as a shorthand notation for $\tau_*(\partial_*(p))$.

gives us the following derivation:

$$\begin{aligned}
\xi &= X_1 \parallel \ldots \parallel X_n \\
&= X_1 \between (X_2 \parallel \ldots \parallel X_n) + \ldots \\
&\qquad + X_n \between (X_1 \parallel \ldots \parallel X_{n-1}) + (X_1 \mid X_2 \mid \ldots \mid X_n) \\
&= \sum_{i \in I_{X_1}} a_i.(\xi_i \parallel X_2 \parallel \ldots \parallel X_n) + \ldots \\
&\qquad + \sum_{i \in I_{X_n}} a_i.(\xi_i \parallel X_1 \parallel \ldots \parallel X_{n-1}) + \mathbf{1}_{\{X_1,\ldots,X_n\}} \\
&= \sum_{V \in \xi} \sum_{i \in I_V} a_i.(\xi_i \xi - \{V\}) + \mathbf{1}_\xi.
\end{aligned}$$

Introducing a fresh variable $\overline{P}(\xi)$ for each possible multiset ξ, we obtain the following equivalent infinite recursive specification.

$$\overline{P}(\xi) = \sum_{V \in \xi} \sum_{i \in I_V} a_i.\overline{P}(\xi_i \xi - \{V\}) + \mathbf{1}_\xi.$$

Now that we have an indication of the relationship between the process P and suitable contents of the bag B, we propose the following equation:

$$\overline{P}(\xi) = \sum_{V \in \xi} \left[\hat{V} \between B_{\xi - \{V\}} \right]_* + \mathbf{1} \mid B_\xi. \tag{1}$$

Equation 1 expresses the relationship between a state in a basic parallel process, given by a parallel composition of variables, and our regular process communicating with a bag. Given that X is the initial variable of E, we can instantiate the general case and use the definition of B and the axioms of Table 3 to show that

$$\begin{aligned}
\overline{P}(X) &= \left[\hat{X} \between B_\emptyset \right]_* + \mathbf{1} \mid B_X \\
&= \left[\hat{X} \between B + B \between \hat{X} \right]_* + \mathbf{1} \mid (!_o X.\mathbf{1} + \mathbf{1}_X \mid B_\emptyset) \\
&= \left[\hat{X} \between B + B \between \hat{X} \right]_* + \mathbf{1}_X \mid B_\emptyset \\
&= \left[\hat{X} \between B + B \between \hat{X} \right]_* + \hat{X} \mid B_\emptyset \\
&= \left[\hat{X} \parallel B \right]_*.
\end{aligned}$$

So, we define $Q = \hat{X}$ and we show that $P = [\hat{X} \parallel B_\emptyset]_*$. This means that we have to prove equation 1 for any multiset of variables ξ.

But first we prove a lemma and corollary relating the definition of the conditional process to communication with the partially forgetful bag.

Lemma 4.2 *For all sets or multisets of variables ξ it holds that $\mathbf{1} \mid B_\xi = \mathbf{1}_\xi$.*

Proof By induction over the contents of ξ.

(i) If $\xi = \emptyset$, then $\mathbf{1} \mid B_\emptyset = \mathbf{1} \mid B = \mathbf{1} = \mathbf{1}_\emptyset$.

(ii) If $\xi = X\xi'$, then
   (a) if $X \in \mathcal{V}^{+\mathbf{1}}$, then $\mathbf{1} \mid B_{X\xi'} = \mathbf{1} \mid (!_o X.\mathbf{1} + \mathbf{1} \parallel B_{\xi'}) = \mathbf{1} \mid !_o X.\mathbf{1} + \mathbf{1} \mid B_{\xi'} = \mathbf{1} \mid B_{\xi'}$. Because, by induction hypothesis, $\mathbf{1} \mid B_{\xi'} = \mathbf{1}_{\xi'}$, we have that $\mathbf{1} \mid B_\xi = \mathbf{1}_{\xi'}$ given that $X \in \mathcal{V}^{+\mathbf{1}}$ and therefore $\mathbf{1} \mid B_\xi = \mathbf{1}_\xi$.
   (b) if $X \notin \mathcal{V}^{+\mathbf{1}}$, then $\mathbf{1} \mid B_{X\xi'} = \mathbf{1} \mid (!_o X.\mathbf{1} \parallel B_{\xi'}) = \mathbf{1} \mid !_o X.\mathbf{1} \mid B_{\xi'} = \mathbf{0} = \mathbf{1}_\xi$ because ξ contains the non-transparent variable X. □

Corollary 4.3 *For all sets or multisets of variables ξ and every variable X it holds that $\mathbf{1}_X \mid B_{\xi - \{X\}} = \mathbf{1}_\xi$.*

Proof By Lemma 4.2, we have that $\mathbf{1}_X \mid B_{\xi-\{X\}} = \mathbf{1} \mid B_X \mid B_{\xi-\{X\}}$. By the definition of B, it follows that $B_X \mid B_{\xi-\{X\}} = B_\xi$. Therefore, again by Lemma 4.2, $\mathbf{1}_X \mid B_{\xi-\{X\}} = \mathbf{1} \mid B_\xi = \mathbf{1}_\xi$. □

Now we prove the following lemma, which is crucial for the main proof. This lemma expresses that if the finite control is at a point where it can choose a variable from the bag, it does not make the actual choice. The choice is determined by the first action that is performed by a candidate variable. It also shows that when this has happened, this particular variable has also been removed from the bag.

Lemma 4.4 *For any non-empty multiset ξ contained in a bag, it holds that $\tau.\,[\mathrm{Ctrl} \parallel B_\xi]_* = \tau.\sum_{V\in\mathcal{V}} \left[\hat{V} \parallel B_{\xi-\{V\}}\right]_*$.*

Proof We use the following definitions: $C = \partial_*(\mathrm{Ctrl} \parallel B_\xi)$, $Y_V = \partial_*(\hat{V} \parallel B_{\xi-\{V\}})$, and $X_V = \text{⁉}_i V.C + Y_V$ for all $V \in \mathcal{V}$. Let us now consider C:

$$\begin{aligned}
C &= \partial_*(\sum_{V\in\mathcal{V}} ?_o V.(!_i V.\mathrm{Ctrl} + \hat{V}) \parallel B) \\
&= \sum_{V\in\mathcal{V}} \text{⁉}_o V.\partial_*((!_i V.\mathrm{Ctrl} + \hat{V}) \parallel B_{\xi-\{V\}}) \\
&= \sum_{V\in\mathcal{V}} \text{⁉}_o V.\partial_*(!_i V.\mathrm{Ctrl} \between B_{\xi-\{V\}} + \hat{V} \between B_{\xi-\{V\}} + \hat{V} \mid B_{\xi-\{V\}}) \\
&= \sum_{V\in\mathcal{V}} \text{⁉}_o V.\big(\text{⁉}_i V.\partial_*(\mathrm{Ctrl} \parallel B_\xi) + \partial_*(\hat{V} \parallel B_{\xi-\{V\}})\big) \\
&= \sum_{V\in\mathcal{V}} \text{⁉}_o V.(\text{⁉}_i V.C + Y_V) \\
&= \sum_{V\in\mathcal{V}} \text{⁉}_o V.X_V.
\end{aligned}$$

If we apply the CFAR^b rule on the specification containing $C = \sum_{V\in\mathcal{V}} \text{⁉}_o V.X_V + \mathbf{0}$ and $X_V = \text{⁉}_i V.C + Y_V$ for each $V \in \mathcal{V}$, which forms a cluster of $\{\text{⁉}_i d, \text{⁉}_o d\}$ in this specification, we obtain: $\tau.\tau_*(C) = \tau.\sum_{V\in\mathcal{V}} \tau_*(Y_V)$. Hence,

$$\tau.\,[\mathrm{Ctrl} \parallel B_\xi]_* = \tau.\tau_*(C) = \tau.\sum_{V\in\mathcal{V}} \tau_*(Y_V) = \tau.\sum_{V\in\mathcal{V}} \left[\hat{V} \parallel B_{\xi-\{V\}}\right]_*.$$

□

Now that all prerequisites are in place, we can deal with the main proof which

requires us to prove the following statement:

$$\overline{P}(\xi) \stackrel{?}{=} \sum_{V \in \xi} \left[\hat{V} \between B_{\xi - \{V\}}\right]_* + \mathbf{1} \mid B_\xi$$

First, apply the definition of $\hat{V}$ and get rid of the left merges.

$$= \sum_{V \in \xi} \sum_{i \in I_V} a_i . \left[\mathrm{Push}(\xi_i) \between B_{\xi - \{V\}}\right]_* + \mathbf{1} \mid B_\xi$$

Perform $|\xi_i|$ pushes by repeatedly applying the definition of $\mathrm{Push}(\xi)$.

$$= \sum_{V \in \xi} \sum_{i \in I_V} a_i . \tau^{|\xi_i|} . \left[\mathrm{Ctrl} \between B_{\xi_i \xi - \{V\}}\right]_* + \mathbf{1} \mid B_\xi$$

Remove all but one τ-step that follows a_i or introduce one τ-step if ξ_i is empty and apply Lemma 4.4 on $\tau . \left[\mathrm{Ctrl} \between B_{\xi_i \xi - \{V\}}\right]_*$.

$$= \sum_{V \in \xi} \sum_{i \in I_V} a_i . \tau . \sum_{W \in \xi_i \xi - \{V\}} \left[\hat{W} \between B_{\xi_i \xi - \{V,W\}}\right]_* + \mathbf{1} \mid B_\xi$$

Remove the τ-step and perform expansion on the merge operator and remove the summand $\left[B_{\xi_i \xi - \{V\}} \between \hat{W}\right]_*$ since its left-hand side cannot perform any non-encapsulated action.

$$= \sum_{V \in \xi} \sum_{i \in I_V} a_i . \Big(\sum_{W \in \xi_i \xi - \{V\}} \left(\left[\hat{W} \between B_{\xi_i \xi - \{V,W\}}\right]_* + \hat{W} \mid B_{\xi_i \xi - \{V,W\}} \right) \Big) + \mathbf{1} \mid B_\xi$$

Now, consider the summand $\hat{W} \mid B_{\xi_i \xi - \{V,W\}}$. Since $\hat{W}$ cannot perform any action, only the summand $\mathbf{1}_W$ remains of the specification of $\hat{W}$.

$$= \sum_{V \in \xi} \sum_{i \in I_V} a_i . \Big(\sum_{W \in \xi_i \xi - \{V\}} \left(\left[\hat{W} \between B_{\xi_i \xi - \{V,W\}}\right]_* + \mathbf{1}_W \mid B_{\xi_i \xi - \{V,W\}} \right) \Big) + \mathbf{1} \mid B_\xi$$

Because $\mathbf{1}_W \mid B_{\xi_i \xi - \{V,W\}} = \mathbf{1}_{\xi_i \xi - \{V\}}$ by Corollary 4.3 and $\mathbf{1}_{\xi_i \xi - \{V\}}$ does not depend on W, we can move it outside of the summation.

$$= \sum_{V \in \xi} \sum_{i \in I_V} a_i . \Big(\sum_{W \in \xi_i \xi - \{V\}} \left[\hat{W} \between B_{\xi_i \xi - \{V,W\}}\right]_* + \mathbf{1}_{\xi_i \xi - \{V\}} \Big) + \mathbf{1} \mid B_\xi$$

Use the definition of $\overline{P}(\xi_i \xi - \{V\})$ and apply Lemma 4.2 on $\mathbf{1} \mid B_\xi$.

$$= \sum_{V \in \xi} \sum_{i \in I_V} a_i . \overline{P}(\xi_i \xi - \{V\}) + \mathbf{1}_\xi .$$

This concludes our proof that there exists a recursive specification over CA that, in parallel with a partially forgetful bag, is equivalent to a basic parallel process P. □

A corollary of this theorem strengthens the result found in [8], that basic parallel processes have finite branching. In fact, the branching is bounded (i.e. there is a fixed maximum branching for all reachable states).

Corollary 4.5 *Every basic parallel process has bounded branching.*

Proof By Theorem 4.1 there exists a regular process Q (given by a finite guarded recursive specification over MA) for every basic parallel process P such that $P = [Q \parallel B_\emptyset]_*$. Because Q is regular, it is boundedly branching. The process B is also boundedly branching. Because the parallel composition leads to the Cartesian product of both boundedly branching components plus the result of communication [5], P is also boundedly branching. □

5 Concluding Remarks

We have proved that every basic parallel process is rooted branching bisimilar to a regular process communicating with a bag. A regular process communicating with a bag can be seen as a parallel pushdown automaton, and so this result extends the result of [11] by adding the process $\mathbf{1}$ and making the internal communication explicit. As a result, we can see the bag as the prototypical basic parallel process. As a corollary, we established that every basic parallel process has bounded branching.

This is in contrast to the situation with context-free processes. We saw in [4] that context-free processes can show unbounded branching. For a context-free process with unbounded branching, we cannot show it is rooted branching bisimilar to a regular process communicating with a stack. We could only show this in contrasimulation. Here, for basic parallel processes, the situation is simpler, and we can establish the full result in rooted branching bisimulation.

The reverse direction, to see if any regular process communicating with a bag is actually a basic parallel process is open as far as we know. Of course, in order to achieve this result, we should allow τ-steps in the definition of basic parallel process, but that is no problem as long as we make sure we retain guardedness. Note that [11] shows the reverse direction is not true in the absence of $\mathbf{1}$, but here, with $\mathbf{1}$, it might still be true.

iIn addition, it is open whether the result of [8] that bisimulation equivalence is decidable for basic parallel processes is still valid with the addition of $\mathbf{1}$.

An interesting extension of basic parallel processes is allowing communication in the definition of processes. Probably, expressive power will increase, but we do not know examples of processes that can be defined with the addition of communication but not without communication.

Finally, note that the addition of $\mathbf{1}$ allows termination exactly when a bag is empty. This check on emptiness is not possible without $\mathbf{1}$. This is different from the situation with a stack, where a check on empty is also possible in an ACP-style language.

Having looked at stacks and bags, it is interesting to look at queues next. Thus, it is interesting to see which set of processes can be realized as a regular process communicating with a queue.

Acknowledgments

The research of Van Tilburg was supported by the project "Models of Computation: Automata and Processes" (nr. 612.000.630) of the Netherlands Organization for Scientific Research (NWO).

References

[1] Aceto, L., W. Fokkink and C. Verhoef, *Structural operational semantics*, in: J. Bergstra, A. Ponse and S. Smolka, editors, *Handbook of Process Algebra*, North-Holland, 2001 pp. 197–292.

[2] Baeten, J., T. Basten and M. Reniers, "Process Algebra: Equational Theories of Communicating Processes," Cambridge University Press, 2008.

[3] Baeten, J. and M. Bravetti, *A ground-complete axiomatization of finite state processes in process algebra*, in: M. Abadi and L. de Alfaro, editors, *Proceedings of CONCUR 2005*, number 3653 in LNCS (2005), pp. 246–262.

[4] Baeten, J., P. Cuijpers and P. v. Tilburg, *A context-free process as a pushdown automaton*, in: *Proceedings of CONCUR 2008*, LNCS (2008), to appear.

[5] Baeten, J. and W. Weijland, "Process Algebra," Cambridge University Press, 1990.

[6] Bergstra, J. and J. Klop, *Process algebra for synchronous communication*, Information and Control **60** (1984), pp. 109–137.

[7] Christensen, S., "Decidability and decomposition in process algebras," Ph.D. thesis, University of Edinburgh (1993).

[8] Christensen, S., Y. Hirshfeld and F. Moller, *Bisimulation equivalence is decidable for basic parallel processes*, in: E. Best, editor, *Proceedings of CONCUR 1993*, number 715 in LNCS (1993), pp. 143–157.

[9] Glabbeek, R. v. and W. Weijland, *Branching time and abstraction in bisimulation semantics*, Journal of the ACM **43** (1996), pp. 555–600.

[10] Moller, F., *The importance of the left merge operator in process algebras*, in: M. Paterson, editor, *Proceedings of ICALP'90*, number 443 in LNCS (1990), pp. 752–764.

[11] Moller, F., *Infinite results*, in: U. Montanari and V. Sassone, editors, *Proceedings of CONCUR '96*, number 1119 in LNCS (1996), pp. 195–216.

[12] Vaandrager, F., *Verification of two communication protocols by means of process algebra*, Technical Report CS-R8608, CWI, Amsterdam (1986).

On convergence-sensitive bisimulation and the embedding of CCS in timed CCS

Roberto M. Amadio [1,2]

Université Paris Diderot

Abstract

We propose a notion of convergence-sensitive bisimulation as a suitable semantic framework for a fully abstract embedding of untimed processes into timed ones.

Keywords: Bisimulation, convergence, timed CCS.

1 Introduction

A first motivation for this work is to build a notion of convergence-sensitive bisimulation from first principles, specifically from the notions of *internal reduction* and of (static) *context*. A second motivation is to understand how asynchronous/untimed behaviours can be embedded fully abstractly into synchronous/timed ones. Because the notion of convergence is very much connected to the notion of time, it seems that a convergence-sensitive bisimulation should find a natural application in a synchronous/timed context. Thus, in a nutshell, we are looking for an 'intuitive' semantic framework that spans both untimed/asynchronous and timed/synchronous models.

For the sake of simplicity we will place our discussion in the well-known framework of (timed) CCS. We assume the reader is familiar with CCS [10]. Timed CCS (TCCS) is a 'timed' version of CCS whose basic principle is that *time passes exactly when no internal computation is possible*. This notion of 'time' is inspired by early work on the Esterel synchronous language [3], and it has been formalised in various dialects of CCS [14,12,6]. Here we shall follow the formalisation in [6].

As in CCS, one models the internal computation with an action τ while the passage of (discrete) time is represented by an action tick that implicitly synchronizes

[1] PPS, UMR-7126. Work partially supported by ANR-06-SETI-010-02.

[2] Email: amadio@pps.jussieu.fr

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

$$\frac{}{a.P \xrightarrow{a} P} \qquad \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{(P \mid Q) \xrightarrow{\tau} (P' \mid Q')}$$

$$\frac{P \xrightarrow{\alpha} P'}{(P \mid Q) \xrightarrow{\alpha} (P' \mid Q)} \qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\frac{A(\mathbf{a}) = P}{A(\mathbf{b}) \xrightarrow{\tau} [\mathbf{b}/\mathbf{a}]P} \qquad \frac{P \xrightarrow{\alpha} P'}{P \triangleright Q \xrightarrow{\alpha} P'}$$

$$\frac{}{0 \xrightarrow{\mathsf{tick}} 0} \qquad \frac{}{a.P \xrightarrow{\mathsf{tick}} a.P}$$

$$\frac{P \overset{\tau}{\not\longrightarrow} \cdot}{P \triangleright Q \xrightarrow{\mathsf{tick}} Q} \qquad \frac{(P_1 \mid P_2) \overset{\tau}{\not\longrightarrow} \cdot \quad P_i \xrightarrow{\mathsf{tick}} Q_i \quad i = 1,2}{(P_1 \mid P_2) \xrightarrow{\mathsf{tick}} (Q_1 \mid Q_2)}$$

$$\frac{P_i \xrightarrow{\mathsf{tick}} Q_i \quad i = 1,2}{P_1 + P_2 \xrightarrow{\mathsf{tick}} Q_1 + Q_2} \qquad \frac{P \xrightarrow{\mu} Q \quad a, \overline{a} \neq \mu}{\nu a\ P \xrightarrow{\mu} \nu a\ Q}$$

Table 1
Labelled transition system

all the processes and moves the computation to the next instant. [3]

In this framework, the basic principle we mentioned is formalised as follows:

$$P \xrightarrow{\mathsf{tick}} \cdot \text{ iff } P \overset{\tau}{\not\longrightarrow} \cdot$$

where we write $P \xrightarrow{\mu} \cdot$ if P can perform an action μ. TCCS is designed so that if P is a process built with the usual CCS operators and P cannot perform τ actions then $P \xrightarrow{\mathsf{tick}} P$. In other terms, CCS processes are *time insensitive*. To compensate for this property, one introduces a new binary operator $P \triangleright Q$, called *else_next*, that tries to run P in the current instant and, if it fails, runs Q in the following instant.

We denote with $\alpha, \beta, \ldots$ the usual CCS actions which are composed of either an internal action τ or of a communication action a. There are countably many communication actions a, each one having a co-action $\overline{a}$. We denote with $\mu, \mu', \ldots$ either an action α or the distinct action tick.

The TCCS processes $P, Q, \ldots$ are specified by the following grammar

$$P ::= 0 \mid a.P \mid P + P \mid P \mid P \mid \nu a\ P \mid A(\mathbf{a}) \mid P \triangleright P\ .$$

We denote with $fn(P)$ the names free in P. We adopt the usual convention that for each thread identifier A there is a unique defining equation $A(\mathbf{b}) = P$ where the parameters $\mathbf{b}$ include the names in $fn(P)$. The related labelled transition system is specified in table 1.

Say that a process is a CCS process if it does not contain the *else_next* operator. The reader can easily verify that:

[3] There seems to be no standard terminology for this action. It is called ϵ in [14], χ in [12], σ in [6], and sometimes 'next' in 'synchronous' languages à la ESTEREL [2].

(1) $P \xrightarrow{\mathsf{tick}} \cdot$ if and only if $P \not\xrightarrow{\tau}$.

(2) If $P \xrightarrow{\mathsf{tick}} Q_i$ for $i = 1, 2$ then $Q_1 = Q_2$. One says that the passage of time is *deterministic*.

(3) If P is a CCS process and $P \xrightarrow{\mathsf{tick}} Q$ then $P = Q$. Hence CCS processes are closed under labelled transitions.
It will be convenient to write $\tau.P$ for $\nu a\ (a.P \mid \overline{a}.0)$ where $a \notin fn(P)$, $\mathsf{tick}.P$ for $0 \triangleright P$, and Ω for the diverging process $\tau.\tau.\ldots$.

Remark 1.1 (1) In the labelled transition system in table 1, the definition of the tick action relies on the τ action and the latter relies on the communication actions $a, a', \ldots$. There is a well known method to give a direct definition of the τ action that does not refer to the communication actions. Namely, one defines (internal) reduction rules such as $(a.P + Q \mid \overline{a}.P' + Q') \to (P \mid P')$ which are applied modulo a suitable structural equivalence.

(2) The labelled transition system in table 1 relies on *negative* conditions of the shape $P \not\xrightarrow{\tau}$. These conditions can be replaced by a condition $\exists L\ \ P \downarrow L$, where L is a finite set of communication actions. This predicate can be defined as follows:

$$\frac{}{0 \downarrow \emptyset} \qquad \frac{}{a.P \downarrow \{a\}} \qquad \frac{P_i \downarrow L_i, \quad i = 1, 2}{(P_1 + P_2) \downarrow L_1 \cup L_2}$$

$$\frac{P \downarrow L}{P \triangleright Q \downarrow L} \qquad \frac{P \downarrow L}{(\nu a\ P) \downarrow L \backslash \{a, \overline{a}\}} \qquad \frac{P_i \downarrow L_i, \quad i = 1, 2 \quad L_1 \cap \overline{L_2} = \emptyset}{(P_1 \mid P_2) \downarrow L_1 \cup L_2}$$

1.1 Signals and a deterministic fragment

As already mentioned, the TCCS model has been inspired by the notion of time available in the Esterel model [4] and its relatives such as SL [5]. These models rely on *signals* as the basic communication mechanism. Unlike a channel, a signal persists within the instant and disappears at the end of instant. It turns out that a signal can be defined recursively in TCCS as:

$$\mathsf{emit}(a) = \overline{a}.\mathsf{emit}(a) \triangleright 0$$

The 'present' statement of SL that either reads a signal and continues the computation in the current instant or reacts to the absence of the signal in the following instant can be coded as follows:

$$\mathsf{present}\ a\ \mathsf{do}\ P\ \mathsf{else}\ Q = a.P \triangleright Q$$

Modulo these encodings, the resulting fragment of TCCS is specified as follows:

$$P ::= 0 \mid \mathsf{emit}(a) \mid \mathsf{present}\ a\ \mathsf{do}\ P\ \mathsf{else}\ P \mid (P \mid P) \mid \nu a\ P \mid A(\mathbf{a})\ .$$

Notice that, unlike in (T)CCS, communication actions have an input or output polarity. The most important property of this fragment is that its processes are *deterministic* [5,1].

1.2 The usual labelled bisimulation

As usual, one can define a notion of *weak transition* as follows:

$$\stackrel{\mu}{\Rightarrow} = \begin{cases} (\stackrel{\tau}{\rightarrow})^* & \text{if } \mu = \tau \\ (\stackrel{\tau}{\rightarrow})^* \circ \stackrel{\mu}{\rightarrow} \circ (\stackrel{\tau}{\rightarrow})^* & \text{otherwise} \end{cases}$$

where the notation X^* stands for the reflexive and transitive closure of a binary relation X. When focusing just on internal reduction, we shall write $\rightarrow$ for $\stackrel{\tau}{\rightarrow}$ and $\Rightarrow$ for $\stackrel{\tau}{\Rightarrow}$. We write $P \rightarrow \cdot$ if $\exists\, P'\ (P \rightarrow P')$, otherwise we say that P has converged and write $P \downarrow$. We write $P \Downarrow$ if $\exists\, Q\ \ (P \Rightarrow Q$ and $Q \downarrow)$. Thus $P \Downarrow$ means that P *may* converge, *i.e.*, there is a reduction sequence to a process that has converged. Because $P \downarrow$ iff $P \stackrel{\mathsf{tick}}{\longrightarrow} \cdot$, we have that $P \Downarrow$ iff $P \stackrel{\mathsf{tick}}{\Rightarrow} \cdot$.

With respect to the notion of weak transition, one can define the usual notion of bisimulation as the largest symmetric relation $\mathcal{R}$ such that if $(P, Q) \in \mathcal{R}$ and $P \stackrel{\mu}{\Rightarrow} P'$ then for some Q', $Q \stackrel{\mu}{\Rightarrow} Q'$ and $(P', Q') \in \mathcal{R}$. We denote with $\approx^u$ the largest labelled bisimulation (u for *usual*). When looking at CCS processes, one may focus on CCS actions (thus excluding the tick action). We denote with $\approx^u_{ccs}$ the resulting labelled bisimulation.

1.3 CCS vs. TCCS

As we already noticed, TCCS has been designed so that CCS can be regarded as a transition closed subset of TCCS. A natural question is whether two CCS processes which are equivalent with respect to an untimed environment are still equivalent in a timed one. For instance, Milner [9] discusses a similar question when comparing CCS to SCCS. [4]

1.3.1 Testing semantics

In the context of TCCS and of a testing semantics, the question has been answered negatively by Hennessy and Regan [6]. For instance, they notice that the processes $P = a.(b + c.b) + a.(d + c.d)$ and $Q = a.(b + c.d) + a.(d + c.b)$ are 'untimed' testing equivalent but 'timed' testing inequivalent. The relevant test is the one that checks that if an action b cannot follow an action a in the current instant then an action b will happen in the following instant just after an action c (process P will not pass this test while process Q does). This remark motivated the authors to develop a notion of 'timed' testing semantics.

1.3.2 Bisimulation semantics

What is the situation with the usual labelled bisimulation semantics recalled in section 1.2? Things are fine for *reactive* processes which are defined as follows.

Definition 1.2 A process P is reactive if whenever $P \stackrel{\mu_1}{\Rightarrow} \cdots \stackrel{\mu_n}{\Rightarrow} Q$, for $n \geq 0$, we have the property that all sequences of τ reductions starting from Q terminate.

[4] The notion of instant in SCCS is quite different from the one considered in TCCS/Esterel. In the former one declares explicitly what each thread does at each instant while in the latter the duration of an instant is the result of an arbitrarily complex interaction among the different threads.

Proposition 1.3 *Suppose P, Q are CCS reactive processes. Then $P \approx^u Q$ if and only if $P \approx^u_{ccs} Q$.*

Proof. Clearly, $\approx^u$ is a CCS bisimulation, hence $P \approx^u Q$ implies $P \approx^u_{ccs} Q$. To show the converse, we prove that $\approx^u_{ccs}$ is a timed bisimulation. So suppose $P \approx^u_{ccs} Q$ and $P \stackrel{\mathsf{tick}}{\Rightarrow} P'$. This means $P \stackrel{\tau}{\Rightarrow} P_1 \stackrel{\mathsf{tick}}{\longrightarrow} P_1 \stackrel{\tau}{\Rightarrow} P'$. Then for some Q_1, $Q \stackrel{\tau}{\Rightarrow} Q_1$ and $P_1 \approx^u_{ccs} Q_1$. Further, because Q_1 is reactive there is a Q_2 such that $Q_1 \stackrel{\tau}{\Rightarrow} Q_2$ and $Q_2 \downarrow$. By definition of bisimulation and the fact that $P_1 \downarrow$, we have that $P_1 \approx^u_{ccs} Q_2$. So for some Q', $Q_2 \stackrel{\tau}{\Rightarrow} Q'$ and $P' \approx^u_{ccs} Q'$. Thus we have shown that there is a Q' such that $Q \stackrel{\mathsf{tick}}{\Rightarrow} Q'$ and $P' \approx^u_{ccs} Q'$. □

Proposition 1.3 fails when we look at *non-reactive* processes. For instance, 0 and Ω are regarded as untimed equivalent but they are obviously timed inequivalent since the second process does not allow time to pass. This example suggests that if we want to extend proposition 1.3 to non-reactive processes, then the notion of bisimulation has to be *convergence sensitive*.

One possibility could be to adopt the usual bisimulation $\approx^u$ on CCS processes. We already noticed that if P is a CCS process and $P \stackrel{\mathsf{tick}}{\longrightarrow} Q$ then $P = Q$. Thus in the bisimulation game between CCS processes, the condition '$P \stackrel{\mathsf{tick}}{\Rightarrow} P'$ implies $Q \stackrel{\mathsf{tick}}{\Rightarrow} Q'$' can be replaced by '$P \Downarrow$ implies $Q \Downarrow$'. The resulting equivalence on CCS processes is not new, for instance it appears in [8] as the so called *stable* weak bisimulation. One may notice that this equivalence has reasonably good congruence properties.

Proposition 1.4 *Suppose $P_1 \approx^u P_2$ and $Q_1 \approx^u Q_2$. Then*

(1) $(P_1 \mid R) \approx^u (P_2 \mid R)$.

(2) *If $P_1, P_2 \downarrow$ then $P_1 \triangleright Q_1 \approx^u P_2 \triangleright Q_2$.*

Proof. First note that we can work with an asymmetric definition of bisimulation where a strong transition is matched by a weak one.

(1) We just check the condition for the tick action. Suppose $(P_1 \mid R) \stackrel{\mathsf{tick}}{\longrightarrow} (P_1' \mid R')$. This entails $P_1 \stackrel{\mathsf{tick}}{\longrightarrow} P_1'$ and $R \stackrel{\mathsf{tick}}{\longrightarrow} R'$. Then $P_2 \stackrel{\tau}{\Rightarrow} P_2''$, $P_2'' \downarrow$, and $P_1 \approx^u P_1''$. Also $P_2'' \stackrel{\mathsf{tick}}{\Rightarrow} P_2'$ and $P_1' \approx^u P_2''$. Finally, we have that $(P_2'' \mid R) \downarrow$ because if they could synchronise on a name a then so could $(P_1 \mid R)$.

(2) There are two cases to consider. If $P_1 \triangleright Q_1 \stackrel{\mathsf{tick}}{\longrightarrow} Q_1$ then $P_2 \triangleright Q_2 \stackrel{\mathsf{tick}}{\longrightarrow} Q_2$. If $P_1 \triangleright Q_1 \stackrel{a}{\rightarrow} P_1'$ because $P_1 \stackrel{a}{\rightarrow} P_1'$ then $P_2 \stackrel{a}{\Rightarrow} P_2'$ and $P_1' \approx^u P_2'$. □

Remark 1.5 The *else_next* operator suffers from the same compositionality problems as the sum operator. For instance, $0 \approx^u \tau.0$ but $0 \triangleright Q = \mathsf{tick}.Q$ while $\tau.0 \triangleright Q \approx^u 0$. As for the sum operator, one may remark that in practice we are interested in a *guarded* form of the *else_next* operator. Namely, the *else_next* operator is only introduced as an alternative to communication actions (the *present* operator discussed in section 1.1 is such an example). Proposition 1.4(2) entails that in this form, the *else_next* operator preserves bisimulation equivalence.

1.3.3 An alternative path

The reader might have noticed that on CCS processes $\approx^u$ *refines* $\approx^u_{ccs}$ by adding may convergence as an observable along with the usual labelled transitions. This is actually the case of all convergence/divergence sensitive bisimulations we are aware of (see, *e.g.*, [15,8]). The question we wish to investigate is what happens if we just take may convergence as an observable without assuming the observability of the labelled transitions? The question can be motivated by both pragmatic and mathematical considerations. On the pragmatic side, one may argue that the normal operation of a timed/synchronous program is to receive inputs at the beginning of each instant and to produce outputs at the end of each instant. Thus, unless the instant terminates, no observation is possible. For instance, the process $(a \mid \Omega)$ could be regarded as equivalent to Ω, while they are distinguished by the usual bisimulation $\approx^u$ on the ground that the labelled transition a is supposed to be directly observable.

On the mathematical side, it has been remarked by many authors that the notion of labelled transition system is not necessarily compelling. Specifically, one would like to define a notion of bisimulation without an *a priori* commitment to a notion of label. To cope with this problem, a well-known approach started in [11] and elaborated in [7] is to look at 'internal' reductions and at a basic notion of 'barb' and then to close under contexts thus producing a notion of 'contextual' bisimulation. However, even the notion 'barb' is not always easy to define and justify (an attempt based on the concept of *bi-orthogonality* is described in [13]). It seems to us that a natural approach which applies to a wide variety of formalisms is to regard convergence (may-termination) as the 'intrinsic' basic observable automatically provided by the internal reduction relation.

1.3.4 Contribution

Following these preliminary considerations, we are now in a position to describe our contribution.

(i) We introduce a notion of contextual bisimulation for (T)CCS whose basic observable (or barb) on CCS processes is the may-termination predicate (section 2).

(ii) We provide various characterisations of this equivalence culminating in one based on a suitable 'convergence-sensitive' labelled bisimulation (section 3).

(iii) We derive from this characterisation that (section 4):
  (a) the embedding of CCS in TCCS is fully abstract (even for non-reactive processes).
  (b) the proposed equivalence coincides with the usual one on reactive processes.
  (c) on non-reactive processes it identifies more processes than the usual *timed* labelled bisimulation $\approx^u$ while it is incomparable with the usual *untimed* CCS bisimulation $\approx^u_{ccs}$.

The development will take place in the context of so called *weak* bisimulation [10] which is more interesting and challenging than *strong* bisimulation.

2 Convergence sensitive bisimulation

We denote with $C, D, \ldots$ one hole *static contexts* specified by the following grammar:

$$C ::= [\,] \mid C \mid P \mid P \mid \nu a C$$

We require that the notion of bisimulation we consider is preserved by the static contexts in the sense of [7].

Definition 2.1 [bisimulation] A symmetric relation $\mathcal{R}$ on processes is a bisimulation if $P\mathcal{R}Q$ implies:

cxt for any static context C, $C[P]\mathcal{R}C[Q]$.

red $P \stackrel{\mu}{\Rightarrow} P'$, $\mu \in \{\tau, \mathsf{tick}\}$ implies $\exists\, Q'$ $(Q \stackrel{\mu}{\Rightarrow} Q'$ and $P'\mathcal{R}Q')$.

We denote with $\approx$ the largest bisimulation.

Remark 2.2 (1) In view of remark 1.1(1), the definition 2.1 of bisimulation does *not* assume the labels $a, a', \ldots$ which correspond to the communication actions. Not only the labels are not considerd in the bisimulation game, but they are not even required in the definition of the τ action. This means that the definition can be directly transferred to more complex process calculi where the definition of the communication actions is at best unclear.

(2) For CCS processes, if $P \xrightarrow{\mathsf{tick}} Q$ then $P = Q$. It follows that in the definition above, the condition **[red]** when $\mu = \mathsf{tick}$ can be replaced by $P \Downarrow$ implies $Q \Downarrow$. This is obviously false for processes including the *else_next* operator; in this case one needs the tick action to observe the behaviour of processes after the first instant, *e.g.*, to distinguish $\mathsf{tick}.a$ from $\mathsf{tick}.b$.

In view of the previous remark, the definition of bisimulation is specialised to CCS processes by simply restricting the condition **[cxt]** to CCS static contexts. We denote with $\approx_{ccs}$ the resulting largest bisimulation.

Next we remark that the observability of a particular kind of 'barb' or 'commitment' is entailed by the observation of convergence.

Definition 2.3 We say that P commits on a, and write $P \Downarrow_a$, if $P \Rightarrow P'$, $P' \downarrow$, and $P' \stackrel{a}{\rightarrow} \cdot$.[5]

Proposition 2.4 *If $P \approx Q$ and $P \Downarrow_a$ then $Q \Downarrow_a$.*

Proof. Suppose $P \Downarrow_a$ and $P \approx Q$. Then $P \Rightarrow P'$, $P' \downarrow$, and $P' \stackrel{a}{\rightarrow} \cdot$. By definition of bisimulation, $Q \Rightarrow Q''$ and $P' \approx Q''$. Moreover, $Q'' \Rightarrow Q'$, $Q' \downarrow$, $Q' \approx P' \approx Q''$. To show that $Q' \stackrel{a}{\rightarrow} \cdot$, consider the context $C = ([\,] \mid \overline{a}.\Omega)$. Then we have $C[P'] \not\Downarrow$, while $C[Q'] \Downarrow$ if and only if $Q' \stackrel{a}{\not\rightarrow} \cdot$. □

Another interesting notion is that of *contextual convergence*.

Definition 2.5 We say that a process P is contextual convergent, and write $P \Downarrow_C$, if $\exists\, C$ $(C[P] \Downarrow)$.

[5] Note that in this definition the process 'commits' on action a only when it has converged.

Clearly, $P \Downarrow$ implies $P \Downarrow_C$ but the converse fails taking, for instance, $(a+b) \mid \overline{a}.\Omega$. Contextual convergence, can be characterised as follows.

Proposition 2.6 *The following conditions are equivalent:*

(1) $P \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} P'$ *and* $P' \downarrow$.

(2) *There is a CCS process* Q *such that* $(P \mid Q) \Downarrow$.

(3) $P \Downarrow_C$.

Proof. (1 $\Rightarrow$ 2) Suppose $P_0 \xrightarrow{\alpha_1} P_1 \cdots \xrightarrow{\alpha_n} P_n$ and $P_n \downarrow$. We build the process Q in (2) by induction on n. If $n = 0$ we can take $Q = 0$. Otherwise, suppose $n > 0$. By inductive hypothesis, there is Q_1 such that $(P_1 \mid Q_1) \Downarrow$. We proceed by case analysis on the first action α_1. If $\alpha_1 = \tau$ take $Q = Q_1$ and if $\alpha_1 = a$ take $Q = \overline{a}.Q_1$.

(2 $\Rightarrow$ 3) Taking the static context $C = [\,] \mid Q$.

(3 $\Rightarrow$ 1) First, check by induction on a static context C that $P \xrightarrow{\tau} \cdot$ implies $C[P] \xrightarrow{\tau} \cdot$. Hence $C[P] \downarrow$ implies $P \downarrow$. Second, show that $C[P] \xrightarrow{\alpha} Q$ implies that $Q = C'[P']$ where C' is a static context and either $P = P'$ or $P \xrightarrow{\alpha'} P'$. Third, suppose $C[P] \xrightarrow{\tau} Q_1 \cdots \xrightarrow{\tau} Q_n$ with $Q_n \downarrow$. Show by induction on n that P can make a series of labelled transitions and reach a process which has converged. □

Remark 2.7 As shown by the characterisation above, the notion of contextual convergence is unchanged if we restrict our attention to contexts composed of CCS processes.

We notice that a bisimulation never identifies a process which is contextual convergent with one which is not while identifying all processes which are not contextual convergent.

Proposition 2.8 (1) *If* $P \approx Q$ *and* $P \Downarrow_C$ *then* $Q \Downarrow_C$.

(2) *If* $P \not\Downarrow_C$ *and* $Q \not\Downarrow_C$ *then* $P \approx Q$.

Proof. (1) If $P \Downarrow_C$ then for some context C, $C[P] \Downarrow$. By condition **[cxt]**, we have that $C[P] \approx C[Q]$ and by condition **[red]** we derive that $C[Q] \Downarrow$. Hence $Q \Downarrow_C$.

(2) We notice that the relation $S = \{(P, Q) \mid P, Q \not\Downarrow_C\}$ is a bisimulation. Indeed: (i) if $P \not\Downarrow_C$ then $C[P] \not\Downarrow_C$, (ii) if $P \Rightarrow P'$ and $P \not\Downarrow_C$ then $P' \not\Downarrow_C$, and (iii) if $P \not\Downarrow_C$ then $P \stackrel{\mathsf{tick}}{\not\Rightarrow} \cdot$. □

3 Characterisation

We characterise the (contextual and convergence sensitive) bisimulation introduced in definition 2.1 by means of a labelled bisimulation. The latter is obtained from the former by replacing condition **[cxt]** with a suitable condition **[lab]** on labelled transitions as defined in table 1.

Definition 3.1 [labelled bisimulation] A symmetric relation $\mathcal{R}$ on processes is a labelled bisimulation if $P \mathcal{R} Q$ implies:

lab if $P \Downarrow_C$ and $P \stackrel{a}{\Rightarrow} P'$ then $Q \stackrel{\alpha}{\Rightarrow} Q'$ and $P' \mathcal{R} Q'$ where $\alpha \in \{a, \tau\}$ and $\alpha = a$ if $P' \Downarrow_C$.

red if $P \stackrel{\mu}{\Rightarrow} P'$, $\mu \in \{\tau, \mathsf{tick}\}$ then $\exists Q'$ $(Q \stackrel{\mu}{\Rightarrow} Q'$ and $P' \mathcal{R} Q')$.

We denote with $\approx^{\ell}$ the largest labelled bisimulation.

Remark 3.2 (1) By remark 2.2, on CCS processes the condition [*red*] when $\mu = \mathsf{tick}$ is equivalent to $P \Downarrow$ implies $Q \Downarrow$. By remark 2.7, the notion of contextual convergence is unaffected if we restrict our attention to CCS processes. This means that, by definition, the (timed) labelled bisimulation restricted to CCS processes is the same as the labelled bisimulation on (untimed) CCS processes.

(2) The predicate of contextual convergence $\Downarrow_C$ plays an important role in the condition **[lab]**. To see why, suppose we replace it with the predicate $\Downarrow$ and assume we denote with $\approx^{\ell\Downarrow}$ the resulting largest bisimulation. The following example shows that $\approx^{\ell\Downarrow}$ is not preserved by parallel composition. Consider

$$P_1 = a.(b + c)\ P_2 = a.b + a.c\ Q = \overline{a}.(d + \Omega)$$

Then $(P_1 \mid Q) \approx^{\ell\Downarrow} (P_2 \mid Q)$ because both processes fail to converge. On the other hand, $(P_1 \mid Q) \mid \overline{d} \not\approx^{\ell\Downarrow} (P_2 \mid Q) \mid \overline{d}$ because the first may converge to $(b + c)$ which cannot be matched by the second process.

(3) One may consider an asymmetric and equivalent definition of labelled bisimulation where strong transitions are matched by weak transitions. To check the equivalence, it is useful to note that $P \not\Downarrow_C$ and $P \stackrel{\alpha}{\rightarrow} P'$ implies $P' \not\Downarrow_C$.

We provide a rather standard proof that bisimulation and labelled bisimulation coincide.

Proposition 3.3 *If $P \approx Q$ then $P \approx^{\ell} Q$.*

Proof. We show that the bisimulation $\approx$ is a labelled bisimulation. We denote with $P \oplus Q$ the internal choice between P and Q which is definable, *e.g.*, as $\tau.P + \tau.Q$. Suppose $P \Downarrow_C$ and $P \stackrel{a}{\Rightarrow} P'$. We consider a context $C = [\ ] \mid T$ where $T = \overline{a}.((b \oplus 0) \oplus c)$ and b, c are fresh names (not occurring in P, Q). We know $C[P] \approx C[Q]$ and $C[P] \Rightarrow (P' \mid (b \oplus 0))$. Thus $C[Q] \Rightarrow (Q' \mid T')$ where either $Q \stackrel{a}{\Rightarrow} Q'$ and $T \stackrel{\overline{a}}{\Rightarrow} T'$ or $Q \Rightarrow Q'$ and $T = T'$.

- Suppose $P' \not\Downarrow_C$. Then $(P' \mid (b \oplus 0)) \not\Downarrow_C$ and, by proposition 2.8, $(Q' \mid T') \not\Downarrow_C$. The latter implies that $Q' \not\Downarrow_C$. By contradiction, suppose $Q' \Downarrow_C$, that is $(Q' \mid R) \Downarrow$. Then $(Q' \mid T') \mid R \mid \overline{T'} \Downarrow$ (contradiction!) where we take $\overline{T'} = \overline{a}$ if $T' = T$ and $\overline{T'} = 0$ otherwise. Hence, $P' \approx Q'$ as required.

- Suppose $P' \Downarrow_C$. If $Q \stackrel{a}{\Rightarrow} Q'$ and $T \stackrel{\overline{a}}{\Rightarrow} T'$ then we show that it must be that $T' = (b \oplus 0)$. This is because if $P' \Downarrow_C$ then $P' \mid (b \oplus 0) \Downarrow_C$ which in turn implies that for some R (not containing the names b or c), $(P' \mid (b \oplus 0) \mid R) \Downarrow_b$. By proposition 2.4, we must have $Q'' = (Q' \mid T') \mid R \Downarrow_b$. Thus T' cannot be 0 and it cannot be $(b \oplus 0) \oplus c$, for otherwise $Q'' \Downarrow_c$ which cannot be matched by $(P' \mid (b \oplus 0) \mid R)$. Further, we have $P' \mid (b \oplus 0) \stackrel{\tau}{\rightarrow} P' \mid 0\ (= P')$. So $(Q' \mid (b \oplus 0)) \stackrel{\tau}{\Rightarrow} (Q' \mid T'')$ and $P' \approx (Q' \mid T'')$. The latter entails that $T'' = 0$.

On the other hand, we show that $Q \stackrel{\tau}{\Rightarrow} Q'$ and $T = T'$ is impossible. Reasoning as above, we have $(P' \mid (b \oplus 0) \mid R) \Downarrow_b$. But then if $(Q' \mid T) \mid R \Downarrow_b$ we shall also have $(Q' \mid T) \mid R \Downarrow_c$. □

The following lemma relates contextual convergence and commitment to labelled bisimulation (cf. the similar propositions 2.8 and 2.4).

Lemma 3.4 (1) *If $P \approx^\ell Q$ and $P \Downarrow_C$ then $Q \Downarrow_C$.*

(2) *If $P \not\Downarrow_C$ and $Q \not\Downarrow_C$ then $P \approx^\ell Q$.*

(3) *If $P \approx^\ell Q$ and $P \Downarrow_a$ then $Q \Downarrow_a$.*

PROOF. (1) By proposition 2.6, if $P \Downarrow_C$ then $P \stackrel{\alpha_1}{\longrightarrow} \cdots \stackrel{\alpha_n}{\longrightarrow} P'$ and $P' \downarrow$. By definition of labelled bisimulation we should have $Q \stackrel{\alpha_1}{\Rightarrow} \cdots \stackrel{\alpha_n}{\Rightarrow} Q'$ and $P' \approx^\ell Q'$. Then $P' \stackrel{\mathsf{tick}}{\Rightarrow} \cdot$ entails $Q' \stackrel{\mathsf{tick}}{\Rightarrow}$, and therefore $Q \Downarrow_C$.

(2) By proposition 2.8, $P, Q \not\Downarrow_C$ implies $P \approx Q$, and by proposition 3.3 we conclude that $P \approx^\ell Q$.

(3) The statement is reduced to check that $P \approx^\ell Q$, $P, Q \downarrow$, and $P \stackrel{a}{\to} P'$ implies $Q \stackrel{a}{\to} Q'$. By definition of $\approx^\ell$, we know that $Q \stackrel{\alpha}{\Rightarrow} Q'$, $P' \approx^\ell Q'$, $\alpha \in \{a, \tau\}$, and $\alpha = a$ if $P' \Downarrow_C$. Now $\alpha = \tau$ leads to a contradiction: if $\alpha = \tau$ then $Q = Q'$, hence $P' \Downarrow_C$, and therefore $\alpha = a$. □

Proposition 3.5 *If $P \approx^\ell Q$ then $P \approx Q$.*

PROOF. We show that labelled bisimulation is preserved by static contexts. In view of remark 3.2(3), we shall work with an asymmetric definition of bisimulation. With respect to this definition, we show that the following relations are labelled bisimulations:

$$\{(\nu a\ P, \nu a\ Q) \mid P \approx^\ell Q\} \cup \approx^\ell , \qquad \{(P \mid R, Q \mid R) \mid P \approx^\ell Q\} \cup \approx^\ell \ .$$

The case for restriction is a routine verification so we focus on parallel composition. Suppose $(P \mid R) \stackrel{\mu}{\to} \cdot$. We proceed by case analysis.

- $(P \mid R) \stackrel{\alpha}{\to} (P \mid R')$ because $R \stackrel{\alpha}{\to} R'$. Then $(Q \mid R) \stackrel{\alpha}{\to} (Q \mid R')$.

- $(P \mid R) \stackrel{\mathsf{tick}}{\longrightarrow} (P' \mid R')$ because $P \stackrel{\mathsf{tick}}{\longrightarrow} P'$ and $R \stackrel{\mathsf{tick}}{\longrightarrow} R'$. Then $Q \stackrel{\tau}{\Rightarrow} Q_1 \stackrel{\mathsf{tick}}{\longrightarrow} Q_2 \stackrel{\tau}{\Rightarrow} Q'$, $P \approx^\ell Q_1$, and $P' \approx^\ell Q'$. By proposition 3.4(3), we derive that $(Q_1 \mid R) \stackrel{\mathsf{tick}}{\longrightarrow} (Q_2 \mid R')$ and thus conclude that $(Q \mid R) \stackrel{\mathsf{tick}}{\Rightarrow} (Q' \mid R')$.

- Suppose $(P \mid R) \Downarrow_C$ and $(P \mid R) \stackrel{a}{\to} (P' \mid R)$ because $P \stackrel{a}{\to} P'$. Then $P \Downarrow_C$ and therefore $Q \stackrel{\alpha}{\Rightarrow} Q'$, $\alpha \in \{a, \tau\}$, and $P' \approx^\ell Q'$. If $P' \Downarrow_C$ then $\alpha = a$ and if $P' \not\Downarrow_C$ then $Q' \not\Downarrow_C$ hence $(P' \mid R) \approx^\ell (Q' \mid R)$ by lemma 3.4.

- Suppose $(P \mid R) \stackrel{\tau}{\to} (P' \mid R)$ because $P \stackrel{\tau}{\to} P'$. Then $Q \stackrel{\tau}{\Rightarrow} Q'$ and $P' \approx^\ell Q'$.

- Suppose $(P \mid R) \stackrel{\tau}{\to} (P' \mid R')$ because $P \stackrel{a}{\to} P'$ and $R \stackrel{\overline{a}}{\to} R'$. If $P, P' \Downarrow_C$ then $Q \stackrel{a}{\Rightarrow} Q'$ and $P' \approx^\ell Q'$. If $P \Downarrow_C$ and $P' \not\Downarrow_C$ then $Q \stackrel{\alpha}{\Rightarrow} Q'$, $\alpha \in \{a, \tau\}$, and $P' \approx^\ell Q'$. But then $(P' \mid R), (Q' \mid R) \not\Downarrow_C$, and we apply lemma 3.4. If $P \not\Downarrow_C$ then $Q \not\Downarrow_C$ and

therefore $(Q \mid R) \not\Downarrow_C$, and we apply again lemma 3.4. □

As a first application of the characterisation we check that bisimulation is preserved be the *else_next* operator in the sense of proposition 1.4(2).

Corollary 3.6 *Suppose $P_1 \approx P_2$, $P_1, P_2 \downarrow$, and $Q_1 \approx Q_2$. Then $P_1 \triangleright Q_1 \approx P_2 \triangleright Q_2$.*

Proof. There are two cases to consider. If $P_1 \triangleright Q_1 \xrightarrow{\mathsf{tick}} Q_1$ then $P_2 \triangleright Q_2 \xrightarrow{\mathsf{tick}} Q_2$. If $P_1 \triangleright Q_1 \xrightarrow{a} P_1'$ because $P_1 \xrightarrow{a} P_1'$ then $P_2 \stackrel{\alpha}{\Rightarrow} P_2'$, $P_1' \approx^{\ell} P_2'$, and $\alpha \in \{\tau, a\}$. We note that it must be that $\alpha = a$. Indeed, if $\alpha = \tau$ then since $P_2 \downarrow$ we must have $P_2' = P_2$ and $P_1' \Downarrow_C$. The latter forces $\alpha = a$ which is a contradiction. □

4 Embedding CCS in TCCS

In this section we collect some easy corollaries of the characterisation. First, we remark that two CCS processes are bisimilar when observed in an untimed/asynchronous environment if and only if they are bisimilar in a timed/synchronous environment.

Proposition 4.1 *Suppose P, Q are CCS processes. Then $P \approx Q$ if and only if $P \approx_{ccs} Q$.*

Proof. By propositions 3.3 and 3.5 we know that $\approx = \approx^{\ell}$. By remark 3.2(1), the labelled bisimulation on untimed processes coincides with the restriction to CCS processes of the timed labelled bisimulation. □

Second we compare the notion of convergence-sensitive bisimulation we have introduced with the usual one we have recalled in the section 1.2. It turns out that all the notions collapse on reactive processes.

Proposition 4.2 *Suppose P, Q are reactive processes. Then $P \approx Q$ if and only if $P \approx^{u} Q$.*

Proof. We know that $\approx = \approx^{\ell}$. Reactive processes are closed under labelled transitions and on reactive processes the conditions that define labelled bisimulation coincide with the ones for the usual bisimulation. □

The situation on non-reactive processes is summarised as follows where all implications are strict.

Proposition 4.3 *Suppose P, Q are processes.*

(1) *If $P \approx^{u} Q$ then $P \approx Q$.*

(2) *If moreover P and Q are CCS processes then $P \approx^{u} Q$ implies both $P \approx^{u}_{ccs} Q$ and $P \approx Q$.*

Proof. (1) The clauses in the definition of $\approx^{u}$ imply directly those in the definition of the labelled bisimulation that characterises $\approx$ (definition 3.1). To see that the converse fails note that $(a \mid \Omega) \approx \Omega$ while $(a \mid \Omega) \not\approx^{u} \Omega$.

(2) Use (1) and the fact that the clauses in the definition of $\approx^u$ imply directly those in the definition of $\approx^u_{ccs}$. To see that the converse fails use the counter-example in (1) and the fact that $0 \approx^u_{ccs} \Omega$ while $0 \not\approx^u_{ccs} \Omega$. □

5 Conclusion

We have presented a natural notion of contextual and convergence sensitive bisimulation and we have shown that it can be characterised by a variant of the usual notion of labelled bisimulation relying on the concept of contextual convergence. As a direct corollary of this characterisation we have shown that (untimed) CCS processes are embedded fully abstractly into timed ones.

The notion of bisimulation we have introduced just requires the notions of reduction and static context as opposed to previous approaches that additionally assume the notion of 'labelled' transition or the notion of 'barb'. It would be interesting to apply the proposed approach to other contexts where the notion of equivalence is unclear. Another related question is to see what happens if one additionally observes must-convergence (strong normalisation). Note that such a 'must-convergence' bisimulation is finer than the one considered here as it distinguishes $A = \tau.A + \tau.0$ from 0.

References

[1] R. Amadio. The SL synchronous language, revisited. *Journal of Logic and Algebraic Programming*, 70:121-150, 2007.

[2] R. Amadio. A synchronous π-calculus. *Information and Computation*, 205(9):1470–1490, 2007.

[3] G. Berry, L. Cosserat. The Esterel synchronous programming language and its mathematical semantics. INRIA technical report 842, Sophia-Antipolis, 1988.

[4] G. Berry and G. Gonthier. The Esterel synchronous programming language. *Science of computer programming*, 19(2):87–152, 1992.

[5] F. Boussinot and R. De Simone. The SL synchronous language. *IEEE Trans. on Software Engineering*, 22(4):256–266, 1996.

[6] M. Hennessy, T. Regan. A process algebra of timed systems. *Information and Computation*, 117(2):221-239, 1995.

[7] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437-486, 1995.

[8] M. Lohrey, P. D'Argenio, and H. Hermanns: Axiomatising Divergence. In Proc. ICALP, SLNCS 2380:585-596, 2002.

[9] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3):267–310, 1983.

[10] R. Milner. Communication and Concurrency. Prentice-Hall, 1989.

[11] R. Milner and D. Sangiorgi. Barbed bisimulation. In Proc. *ICALP*, SLNCS 623:685–695, 1992.

[12] X. Nicolin, J. Sifakis. The algebra of timed processes (ATP): theory and application. *Information and Computation*, 114(1):131-178, 1994.

[13] J. Rathke, V. Sassone and P. Sobocinski. Semantic barbs and biorthogonality. In Proc. FoSSaCS 2007, SLNCS 4423:302-316, 2007.

[14] W. Yi. A calculus of real time systems. PhD thesis. Chalmers University, 1991.

[15] D. Walker. Bisimulation and divergence. Information and Computation, 85:202-241, 1990.

Adding Branching to the Strand Space Model

Sibylle Fröschle[1]

Department für Informatik
Universität Oldenburg
Oldenburg, Germany

Abstract

The strand space model is one of the most successful and widely used formalisms for analysing security protocols. This might seem surprising given that the model is not able to reflect choice points in a protocol execution: the key concept in the strand space model is that of a bundle, which models exactly one possible execution of a security protocol. Inspired by the branching processes of Petri nets, we show that branching can be introduced into the strand space model in a very natural way: bundles can be generalized to branching bundles, which are able to capture several conflicting protocol executions. Our investigations of the theory of branching bundles will motivate the concept of symbolic branching bundles, and culminate in the result that every protocol has a strand space semantics in terms of a largest symbolic branching bundle. We hope our results provide a strong theoretical basis for comparing models and providing process calculi semantics in security protocol analysis. Altogether our work is related but different to a series of works by Crazzolara and Winskel. Throughout we will profit from a close relationship of the strand space model to event structures, which has already been pointed out by these authors.

Keywords: Models for security protocol analysis, strand spaces, event structures, branching processes

1 Introduction

The strand space model [5] is one of the most successful and widely used formalisms for analysing security protocols. For example, it has been employed to verify security properties by hand, to give formal semantics to protocol logics, and as the underlying model of model-checking tools (c.f. [7]). In spite of this success two points of criticism have been brought against it: one is that, in contrast to models based on multiset rewriting, it is an ad hoc model rather than rooted in a rich theory. The second is that it is not able to reflect aspects of branching such as choice points in a protocol execution. To explain the latter we recall that the central concept of the strand space model is that of the *bundle*. A bundle models exactly one snapshot of a protocol execution: a set of strands represents the sessions that have occurred so far while a relation between the send and receive events of the strands describes how messages have flowed between them.

[1] Email: froeschle@informatik.uni-oldenburg.de

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

Both points of criticism have been countered by results of Crazzolara and Winksel [2,1,3]. On the one hand, they have shown that the strand space model is closely related to event-based models for concurrency such as event structures. On the other hand, aiming to obtain compositional event-based semantics for protocol languages, they have shown how the strand space model can be extended by a notion of conflict [3]. Their notion of conflict is introduced at the level of strand spaces, which are conceptually a level lower than bundles: a strand space fixes all the sessions that are to be considered in the protocol analysis; it is the space from which bundles are 'carved out'.

In this paper our thesis is that branching can be introduced into the strand space model very directly at the conceptual level of bundles. The idea is to generalize bundles to *branching bundles* in the same way as in Petri net theory branching processes generalize Petri net processes. Petri net branching processes [4] are introduced as a formalization of an initial part of a run of a Petri net, which can include conflicting choices. They come with a very satisfying theory: the branching processes of a Petri net form a complete lattice (modulo isomorphism) with respect to a natural notion of approximation. The largest element of this lattice captures all possible behaviour of the Petri net, and is called its unfolding.

In this paper we wish to investigate whether protocols have as satisfying a theory of branching bundles. If every protocol P had indeed a largest branching bundle, this branching bundle would capture all possible behaviour of P, and would thus provide a natural strand space semantics. This could provide a strong theoretical basis for comparing models and providing semantics for protocol languages. Our contributions are as follows:

(1) We show that bundles can indeed be generalized to *branching bundles* in a very natural way. Branching bundles are able to capture several conflicting protocol executions.

(2) We investigate the theory of branching bundles. We find that every branching bundle can be viewed as a labelled prime event structure. This will motivate a notion of computation state for branching bundles in terms of sub-bundles, and a transition relation between them. Following the approach of [4] we will investigate whether the branching bundles of a protocol form a complete lattice with respect to a natural notion of approximation. We will however obtain a negative result: the branching bundles of a protocol do not even form a complete partial order.

(3) By analysing this negative result we will, however, be led to a notion of *symbolic* branching bundle. We will obtain that the symbolic branching bundles of a protocol indeed form a complete lattice (modulo isomorphism). Thus, every protocol can be given a strand space semantics in terms of a largest symbolic branching bundle. We will motivate that this semantics is suitable for most situations in security protocol analysis.

In the following section we will introduce the strand space model using variations on the original definition of [2] and [6]. The remainder of the paper is structured according to the above contributions. All proofs can be found in the full version of the paper.

2 The Strand Space Model

Graph Terminology
A *labelled (directed) graph* is a tuple (E, K, L, l) where E is the set of nodes, which in our context will depict *events*, $K \subseteq E \times E$ is the set of edges, L is the set of labels, and $l : E \to L$ is a labelling function that assigns a label to every node. When L is clear from the context we will often keep it implicit for notational simplicity.

A labelled graph (E, K, L, l) is *totally ordered* if there is a total ordering $e_1 e_2 \ldots$ of the elements of E such that $(e_i, e_j) \in K$ if and only if $j = i + 1$. A labelled graph (E, K, L, l) is a *labelled tree* if K is acyclic and there is no backwards branching, i.e., if $(e', e) \in K$ and $(e'', e) \in K$ then $e' = e''$. A *branch* of a tree is a possibly empty, finite or infinite sequence $e_1 e_2 \ldots$ of elements of E such that $(e_i, e_{i+1}) \in K$ for all indices i. A *labelled bi-graph* is a tuple $(E, \Rightarrow, \rightarrow, L, l)$ such that both $(E, \Rightarrow, L, l)$ and $(E, \rightarrow, L, l)$ are labelled graphs.

Message Algebra
In the following we assume that messages are modelled by a message algebra. The results we present here are independent of the actual structure of this algebra. We only assume a set of messages *Mesg*, a set of atomic messages *AMesg*, from which *Mesg* is built up from, and a binary relation $\sqsubseteq$ on *Mesg* that says when one message is contained in another. Messages and atomic messages may contain variables. A message is ground if it does not contain any variables. We denote the set of ground messages by *GMesg*.

Actions, Roles, and Protocols
In a protocol execution, principals can either send or receive messages. If a message is sent then it can contain data that have just been freshly generated such as nonces. This gives rise to the following set of *actions*:

$$\begin{aligned} Act = \{ + \mathit{fresh}\ \mathcal{N}\ \mathit{in}\ M \mid M \in \mathit{Mesg}\ \&\ \mathcal{N} \subseteq \mathit{AMesg}\ \&\ \forall N \in \mathcal{N}, N \sqsubseteq M\} \\ \cup \{ - M \mid M \in \mathit{Mesg}\}. \end{aligned}$$

In an action of the form '$+$ *fresh* $\mathcal{N}$ *in* M', '$+$' indicates that message M is thought to be sent while $\mathcal{N}$ specifies which elements of M are thought to be freshly generated. We assume that only atomic messages can be freshly generated. In an action of the form '$- M$', '$-$' indicates that message M is thought to be received. Given an action $A \in Act$ of either of the two forms we use $\mathit{mesg}(A)$ to depict M, $\mathit{sign}(A)$ to depict '$+$', or '$-$' respectively. If $\mathit{sign}(A) = +$ we will further use $\mathit{fresh}(A)$ to depict $\mathcal{N}$. A *ground action* is an action that does not contain any variables. We denote the set of ground actions by *GAct*. In the context of a labelled graph with label set *GAct* we will carry over the previous concepts for actions to the events of the graph in the obvious way. A *trace* is a finite sequence of ground actions.

A *role* defines the actions a principal can perform in a protocol session. Formally, a role is a finite sequence of actions $R = A_1 \ldots A_n$ such that

R1 for all $i \in [1, n]$, for all $N \in \mathit{fresh}(A_i)$
  (a) N is a variable, and

(b) A_i is the first action that contains N: $\forall j < i,\ N \not\sqsubseteq \mathit{mesg}(A_j)$.

Axiom (R1) makes sure that we cannot specify a constant to be freshly generated, and that variables that represent data to be freshly generated at some action cannot occur in previous actions.

A *protocol* is a finite set of *roles* $P = \{R_i\}_{i \in r}$ where $r \in \mathbb{N}$.

Intruder Model

The power of the Intruder is typically modelled by two ingredients: the set of messages initially known to the Intruder such as all public keys and his own private key; and a set of *Intruder roles*, which specify the Intruder's basic elements of attack such as decrypting a message with a key that he has already obtained. (Intruder roles are originally called *parametric Intruder traces* [5].) Similarly to protocol roles, Intruder roles are essentially sequences of signed messages, where '+' denotes output and '−' denotes input.

The results here are independent of the actual format of the Intruder roles. We only assume that an *Intruder theory* is given as a pair $I = (\mathcal{K}_I, \mathcal{R}_I)$ where $\mathcal{K}_I \subseteq \mathit{GMesg}$ is the set of *initial Intruder knowledge* and $\mathcal{R}$ is the set of *Intruder roles*, and that each Intruder role is a finite sequence of actions of the following form:

$$\mathit{IAct} = \{ + M \mid M \in \mathit{Mesg}\} \cup \{ - M \mid M \in \mathit{Mesg}\}$$
$$\cup \{ + M \ \mathit{of\ I\text{-}Knowledge} \mid M \in \mathit{Mesg}\}.$$

We redefine the set of actions Act defined in the previous paragraph to include actions of this form: $\mathit{Act} := \mathit{Act} \cup \mathit{IAct}$.

Strands, Strand Spaces, and Bundles

We now come to the core notions of the strand space model: *strands* and *bundles*. We define these concepts relative to a fixed protocol P.

A *strand* represents an instantiation of a protocol or Intruder role or of a prefix thereof. (We admit prefixes to be able to model incomplete protocol or Intruder sessions, a situation that naturally arises in a snaphot of a protocol execution.) Formally, a *strand* of P is a *totally ordered* labelled graph $s = (E, \Rightarrow, \mathit{GAct}, l)$ such that there is a prefix R of a role of P or $\mathcal{R}_I$ and a ground substitution σ so that, assuming

- $E = \{e_1, \ldots, e_n\}$ with $e_1 \Rightarrow \ldots \Rightarrow e_n$, and
- $R = A_1 \ldots A_m$,

we have

S1 $l(e_1) \ldots l(e_n) = A_1\sigma \ldots A_m\sigma$,

S2 $\forall i \in [1, n]$, if $\mathit{sign}(e_i) = +$ and $n \in \mathit{fresh}(e_i)$ then for all $j < i$, $n \not\sqsubseteq \mathit{mesg}(e_j)$,

S3 $\forall e \in E$, if $l(e)$ is of the form '$+\ m\ \mathit{of\ I\text{-}Knowledge}$' then $m \in \mathcal{K}_I$.

Observe how the axioms ensure that s can indeed be understood as an instantiation of R via σ. We call E the set of *events of* s, denoted by $\mathit{events}(s)$. If an event e has sign '+', we call it a *send event*, and if it has sign '−', a *receive event* respectively.

We say message m *originates on event* e_i if e_i is a send event, $m \sqsubseteq mesg(e_i)$, and for all $j < i$, $m \not\sqsubseteq mesg(e_j)$. Note that Axiom (S2) ensures that when an atomic message is freshly generated at an event then it originates on that event. We call $l(e_1) \ldots l(e_n)$ the *trace* of strand s. We say two strands are *disjoint* if their sets of events are disjoint.

A snapshot of a protocol execution consists of the set of (complete and incomplete) protocol and Intruder sessions that have been executed so far plus information on how the messages flow between the sessions. This leads us to the concept of *strand space.*[2] A *strand space* of P is a pair $B = (S, \rightarrow)$ where S is a set of pairwise disjoint strands of P, and $\rightarrow \subseteq E \times E$ is a relation on the *events* of S, $E = \bigcup_{s \in S} events(s)$. The single-arrow relation is thought to represent the flow of messages. It is clear that we can equivalently regard B as a labelled bi-graph $(E, \Rightarrow, \rightarrow, GAct, l)$, a view we will often adopt. We call E the set of *events of* B, denoted by $events(B)$.

A strand space can contain situations that are counter-intuitive such as a receive event leading to a send event. A snapshot of a protocol execution is modelled by a *bundle*. Formally, a *bundle* of P is a strand space $B = (E, \Rightarrow, \rightarrow, GAct, l)$ of P such that the following axioms are satisfied:

B1 if $e_1 \rightarrow e_2$ then $sign(e_1) = +$, $sign(e_2) = -$, and $mesg(e_1) = mesg(e_2)$,

B2 if $e_1 \rightarrow e_2$ then there is no other e_1' such that $e_1' \rightarrow e_2$,

B3 $\forall e \in E$, if $sign(e) = -$ then there is $e' \in E$ such that $e' \rightarrow e$,

B4 the relation $(\rightarrow \cup \Rightarrow)$ is acyclic,

B5 $\forall e \in E$, $\{e' \mid e' \ (\rightarrow \cup \Rightarrow)^* \ e\}$ is finite,

B6 $\forall e \in E$, if $sign(e) = +$ and $n \in fresh(e)$ we have: n is *uniquely originating* on e: there is no event e' with $e' \neq e$ such that n originates on e'.

Axiom (B1) ensures that messages flow from send events to receive events. Axiom (B2) enforces that an event can receive its message from at most one event. Axiom (B3) guarantees that each receive event is matched up with a send event. Axiom (B4) ensures that the reflexive and transitive closure of $\rightarrow \cup \Rightarrow$ is a partial order, which, as we will explain below, captures *causality*. Axiom (B5) ensures that every event depends on only finitely many previous events. It is necessary in our setting since we allow bundles to contain infinitely many events. Axiom (B6) ensures that if an atomic message is specified to be freshly generated on some event then on any other strand it has to be received before it can be sent.

We denote the relation $\rightarrow \cup \Rightarrow$ by $\prec_1$. $\prec_1$ expresses *immediate causality*: If $e \rightarrow e'$ then e is an immediate cause of e' due to the message flow causality between received messages and sent messages. If $e \Rightarrow e'$ then e is an immediate cause of e' due to the execution order causality within a protocol session. The reflexive and transitive closure of $\prec_1$, denoted by $\preceq$, is a partial order, which captures *causality*.

For every event e of a bundle there is at most one event e' such that $e' \Rightarrow e$, and at most one event e'' such that $e'' \rightarrow e$. If the first exists define $\Rightarrow\text{-}pred(e) = e'$ otherwise define $\Rightarrow\text{-}pred(e) = nil$. If the latter exists define $\rightarrow\text{-}pred(e) = e''$, and $\rightarrow\text{-}pred(e) = nil$ otherwise. Naturally we assume $nil \notin E$.

[2] This notion slightly varies from the standard notion of strand space related to in the introduction.

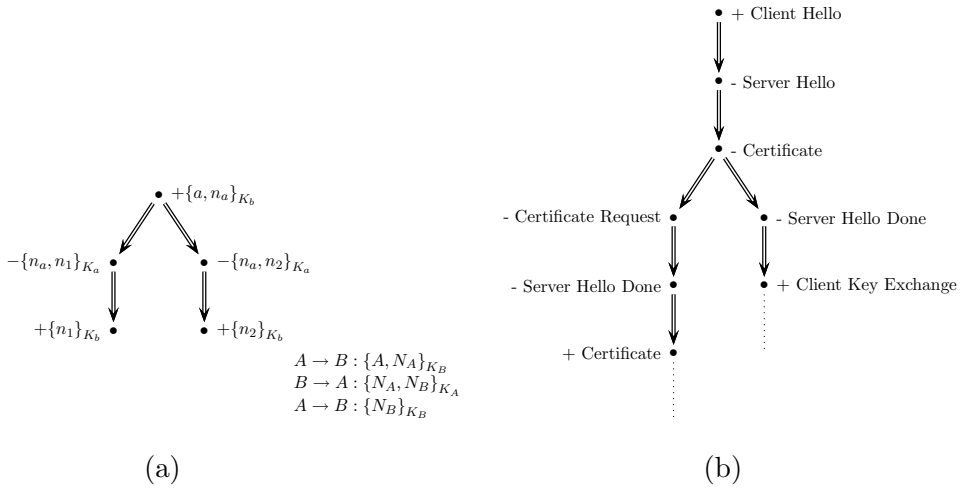


Fig. 1. Sources of branching within a session

3 Branching Bundles

We now define our concept of *branching bundles*. As motivated in the introduction branching bundles should be capable of representing several conflicting protocol executions. To obtain them as a natural generalization of bundles, we will define them as bi-graphs of events labelled by actions of $GAct$. In contrast to bundles we will allow them to contain events that represent conflicting points in a protocol execution. We can distinguish between three situations when two events e_1 and e_2 of a protocol execution should naturally be considered to be in conflict with each other:

(i) e_1 and e_2 belong to different futures of the same session;

(ii) one of e_1 and e_2, say e_1, is a send event that sends as part of its message a freshly generated atomic message n, while the other event, e_2, contradicts unique origination of n: e_2 sends n as part of its message but n has never been received earlier in the session of e_2.

(iii) e_1 and e_2 are causally dependent on two events that are in conflict according to (i) or (ii).

The first situation motivates that a concept of branching bundles must be based on a concept of *branching strands*. So let us analyse in turn what sources of branching there are within a session. When does a session split into different futures?

(i) The receive actions of a protocol specification typically contain variables to be bound to parts of the incoming message. A session with such a receive action will have different futures depending on the received message. (The different futures will, however, be equivalent modulo the value that is bound to the respective input variable.) This situation is depicted in Figure 1(a).

(ii) The protocol specification may contain choice points. For example, the course of the SSL/TLS handshake protocol depends on which method for establishing the pre-master-secret is negotiated at the start, and on whether client authen-

tication is requested by the Server or not. Typically the choice between several options of a protocol will be resolved by received input. On the other hand, to abstract away from detail, we may allow protocols to contain nondeterministic choice. The first situation is depicted in Figure 1(b).

There is yet another source of branching if we take a purely observational view. Say Eavesdropper Eve observes the first two actions of a session, but she cannot tell to which role these actions belong to: according to their format the actions could form an initial part of an instance of role A or of role B. Then in one future of Eve the *observed session* may evolve into an instance of role A, whereas in another future of Eve it will evolve into an instance of role B.

(iii) An *observed* session may have different futures due to ambiguity in the protocol specification.

To include (iii) as a source of branching is a design decision and may seem counter-intuitive at first. To include it seems, however, in the spirit of the strand space model: it is consistent with the fact that if there is ambiguity in the protocol specification then a strand may be interpreted as an instance of several roles. To resolve this type of ambiguity one would need to keep a role identifier at each strand, which would make the model less abstract and technically cumbersome. As we will now see our decision to include (iii) leads to a very simple formalization of branching strand. In the following, let P be a protocol.

Definition 3.1 A *branching strand* of P, abbreviated *b-strand* of P, is a labelled tree $s = (E, \Rightarrow, GAct, l)$ such that all branches of s are strands of P. (Note that this implies that branching strands are trees of finite depth.)

By definition every strand is a b-strand, and every b-strand whose events are totally ordered is a strand. We carry over all the concepts defined for strands in Section 2 in the obvious way. The notation $\Rightarrow$-$pred(e)$ naturally also carries over.

Having defined a notion of b-strands we obtain b-strand spaces in the obvious way. A *b-strand space* of P is a pair $B = (S, \rightarrow)$ where S is a set of disjoint b-strands of P, and $\rightarrow \subseteq E \times E$ is a relation on the events of S, $E = \bigcup_{s \in S} events(s)$. Analogously to strand spaces, we will often view B as a labelled bi-graph $(E, \Rightarrow, \rightarrow, GAct, l)$. The three situations of conflict pinpointed in the beginning of the section give rise to a binary conflict relation on the events of a b-strand space.

Definition 3.2 Let $B = (E, \Rightarrow, \rightarrow, GAct, l)$ be a b-strand space of P. Two *distinct* events e_1, $e_2 \in E$ are *in immediate conflict*, written $e_1 \#_1 e_2$, if

(i) $\Rightarrow$-$pred(e_1) = \Rightarrow$-$pred(e_2)$, or

(ii) $sign(e_1) = +$, and there is $n \in fresh(e_1)$ such that n originates on e_2, or

(iii) the symmetric condition holds.

Two events e_1, $e_2 \in E$ are *in conflict*, written $e_1 \#_B e_2$, if there exist distinct events e_1', $e_2' \in E$ such that $e_1' \#_1 e_2'$ and e_i' $(\Rightarrow \cup \rightarrow)^*$ e_i for $i = 1, 2$. For $e \in E$, we say e is *in self-conflict* if $e \#_B e$.

To be able to interpret a b-strand space as a branching protocol execution we need to impose axioms. Naturally we will adopt Axioms (B1) to (B5) of the def-

inition of bundles. However, Axiom (B6) will be dropped: the axiom on unique origination is clearly not needed when events are allowed to be in conflict. On the other hand, in the presence of conflict a new axiom will be required: we need to make sure that events are never in conflict with any of the events they are causally dependent on. Formally, this gives rise to the following definition.

Definition 3.3 A *branching bundle* of P, abbreviated *b-bundle* of P, is a b-strand space B of P such that Axioms (B1) to (B5) as well as the following axiom hold:

BB No event of B is in self-conflict.

Analogously to bundles, due to Axiom (B4), we can associate a causality relation $\preceq_B$ with each b-bundle B; we carry over all concepts related to $\preceq_B$ from bundles to b-bundles. Due to Axiom (BB), for every b-bundle B, $\#_B$ is irreflexive as well as symmetric; this is what one would expect of a binary conflict relation.

Finally, we show that bundles are exactly those b-bundles where no events are in conflict. This illustrates that b-bundles are indeed the generalization of bundles we have been looking for.

Proposition 3.4 *For every b-strand space B of P, B is a bundle of P if and only if B is a b-bundle with $\#_B = \emptyset$.*

4 Towards a Theory of Branching Bundles

We now investigate whether b-bundles have as satisfying a theory as Petri net branching processes. In Section 4.2 we examine the relationship of b-bundles to event structures. In Section 4.3 we explore whether the b-bundles of a protocol approximate (modulo isomorphism) a largest b-bundle. In preparation, we introduce a notion of sub-b-bundle in Section 4.1, which is analogous to that of Petri net sub-b-processes [4].

In the following, we work as usual relative to a fixed protocol P. Given a b-bundle B of P we will implicitly assume $B = (E, \Rightarrow, \rightarrow, GAct, l)$. We carry this convention over to b-bundles B_1, B_2, and B' in the obvious way; e.g., we assume $B_1 = (E_1, \Rightarrow_1, \rightarrow_1, GAct, l_1)$.

B-bundles come with a notion of *isomorphism* induced by the standard notion for labelled bi-graphs. As usual the relation 'isomorphic' is an equivalence relation. Next to isomorphism a notion of *homomorphism* for b-bundles will be central. A homomorphism h from b-bundle B_1 to b-bundle B_2 formalizes the fact that B_1 can be folded onto a part of B_2. Given an event e of a b-bundle B, we define $\Downarrow e = \{e' \in E \mid e' \ (\Rightarrow)^* \ e\}$.

Definition 4.1 Let B_1 and B_2 be two b-bundles of P. A *homomorphism from B_1 to B_2* is a mapping h from E_1 to E_2 such that

(i) for every $e \in E_1$, $l_1(e) = l_2(h(e))$,

(ii) for every $e \in E_1$, the restriction of h to $\Downarrow_1 e$ is a bijection between $\Downarrow_1 e$ and $\Downarrow_2 h(e)$, and

(iii) for every $e, e' \in E_1$, if $e \rightarrow_1 e'$ then $h(e) \rightarrow_2 h(e')$.

It is easy to show that the composition of two homomorphisms is a homomorphism. If a homomorphism is bijective then the converse of (iii) is also true (using the fact that B_1 and B_2 are b-bundles). Thus, an isomorphism is a bijective homomorphism.

4.1 Sub-b-bundles

We introduce a natural notion of *sub-b-bundle*, which formalizes when a b-bundle is an initial part of another b-bundle.

Definition 4.2 Let B and B' be two b-bundles of P. B' is a *sub-b-bundle of* B if $E' \subseteq E$ and the identity on E' is a homomorphism from B' to B. If B' is a bundle we also say that B' is a *sub-bundle* of B.

In other words, B' is a sub-b-bundle of B if, $E' \subseteq E$, and for every $e \in E'$, (1) $l'(e) = l(e)$, (2) $\Rightarrow'\text{-}pred(e) = \Rightarrow\text{-}pred(e)$, and (3) $\rightarrow'\text{-}pred(e) = \rightarrow\text{-}pred(e)$ (using the fact that B and B' are b-bundles). This shows that B' really is an initial part of B.

We provide a characterization of the sub-b-bundles and sub-bundles of a b-bundle B in terms of *downwards-closed* subsets of E. This will further illustrate the concept of sub-b-bundle but will also be needed in the next section. A subset E' of E is *downwards-closed* if, for all $e_1, e_2 \in E$, if $e_1 \preceq_B e_2$ and $e_2 \in E'$ then $e_1 \in E'$. If B' is a sub-b-bundle of B then E' is clearly downwards-closed. This follows from the observation of the previous paragraph. On the other hand, every downwards-closed set of events determines a sub-b-bundle in a natural way.

Definition 4.3 Let B be a b-bundle, and let E' be a downwards-closed subset of E. The *sub-b-bundle* associated with E', denoted by $sbb(E')$ is defined as $(E', \Rightarrow', \rightarrow', GAct, l')$, where $\Rightarrow'$, $\rightarrow'$, and l' are obtained as the restriction of $\Rightarrow$, $\rightarrow$, and l respectively, to the events in E'.

It is immediate that $sbb(E')$ is indeed a b-bundle. It is clearly a sub-b-bundle by definition. We are now ready to state the characterization.

Proposition 4.4 *Let B be a b-bundle.*

(i) *A b-bundle B' is a sub-b-bundle of B if and only if $B' = sbb(E')$ for some downwards-closed subset E' of E.*

(ii) *A bundle B' is a sub-bundle of B if and only if $B' = sbb(E')$ for some downwards-closed* and conflict-free *subset E' of E.*

4.2 Branching Bundles and Event Structures

A *(labelled prime) event structure* is a tuple $(E, \leq, \#, L, l)$ consisting of a set E of *events*, which are partially ordered by $\leq$, the *causal dependency relation*, a binary, symmetric and irreflexive relation $\# \subseteq E \times E$, the *conflict relation*, a set L of *labels*, and a *labelling function* $l : E \to L$, which assigns a label to each event. Further, the following conditions must be satisfied for all $e, e', e'' \in E$:

E1 $e{\downarrow} = \{e' \mid e' \leq e\}$ is finite,

E2 if $e\#e'$ and $e' \leq e''$ then $e\#e''$.

Axiom (E1) means we only consider discrete processes where an event occurrence depends on finitely many previous events. Axiom (E2) makes sure that each event inherits conflict from the events it is causally dependent on.

Event structures come equipped with a notion of computation state, called *configuration*, and a transition relation between configurations. A *configuration* of an event structure $(E, \leq, \#, L, l)$ is a set $X \subseteq E$, which is

(i) downwards-closed: $\forall e, e' \in E : e' \leq e \;\&\; e \in X \Rightarrow e' \in X$, and

(ii) conflict-free: $\forall e, e' \in X : \neg(e\#e')$.

For two configurations X, X' and an event e we write $X \xrightarrow{l(e)} X'$ when $e \notin X$ and $X' = X \cup \{e\}$. In this way every event structure gives rise to a labelled transition system.

We shall now see that b-bundles are closely related to event structures. The following is straightforward:

(i) Every b-bundle B of P can be viewed as an event structure. This event structure gives a more abstract representation of B in that it abstracts away from the distribution of events over b-strands.

Proposition 4.5 *Let $B = (E, \Rightarrow, \rightarrow, GAct, l)$ be a b-bundle of P. Then $bb2ev(B) := (E, \preceq_B, \#_B, GAct, l)$ is an event structure.*

Just as the configurations of an event structure define its computation states, the sub-bundles of a b-bundle can be considered to define the reachable states of that part of the protocol execution described by the b-bundle. From Section 4.1 we know that the sub-bundles of a b-bundle can be captured in terms of conflict-free and downwards-closed subsets of events. Hence, we obtain:

(ii) There is a one-to-one correspondence between the sub-bundles of B and the configurations of $bb2ev(B)$, given by:

Proposition 4.6 *Let B be a b-bundle of P.*

(i) *If B' is a sub-bundle of B then E' (the set of events of B') is a configuration of $bb2ev(B)$.*

(ii) *If E' is a configuration of $bb2ev(B)$ then $sbb(E')$ is a sub-bundle of B'.*

We can define a transition relation between the sub-bundles of a b-bundle analogously to how this is done for event structures: given a b-bundle B, for two sub-bundles B', B'' of B, and an event $e \in E$, we write $B' \xrightarrow{l(e)} B''$ when $e \notin E'$ and $E'' = E' \cup \{e\}$. Altogether, we have:

(iii) Every b-bundle B induces a labelled transition system, where the states are given by the sub-bundles of B and the transition relation describes how a sub-bundle can evolve into a new one by executing an action. The induced labelled transition system is isomorphic to that induced by $bb2ev(B)$.

4.3 Approximation

Every b-bundle of a protocol P captures an initial part of the behaviour of P. We now wish to investigate whether the b-bundles of P consistently approximate, modulo isomorphism, a largest b-bundle. If every protocol P had indeed a largest b-bundle, this b-bundle would capture all possible behaviour of P, and would thus provide a natural strand space semantics for protocols. Furthermore, in view of the results of the previous section this strand space semantics would come with a notion of computation state in terms of bundles, and a transition relation between them. The induced labelled transition system would give the corresponding interleaving semantics of the protocol, while the protocol would also have an abstract partial order semantics in terms of the induced labelled event structure.

First, we need to define a natural notion of approximation for b-bundles. Intuitively, one b-bundle approximates another when it is, up to isomorphism, an initial part of the other. This can be formalized as follows.

Definition 4.7 Let B_1, B_2 be two b-bundles of P. B_1 approximates B_2, written $B_1 \leq B_2$, if there exists an injective homomorphism from B_1 to B_2.

The following observation justifies the naturalness of this definition:

Proposition 4.8 *Let B_1, B_2 be two b-bundles of P. $B_1 \leq B_2$ if and only if B_1 is isomorphic to a sub-b-bundle of B_2.*

Naturally, approximation is preserved by isomorphism. Thus, $\leq$ can be extended to isomorphism classes of b-bundles. Let $IBB(P)$ denote the set of isomorphism classes of b-bundles of P. As one would expect $\leq$ is a partial order on $IBB(P)$.

Proposition 4.9 $(IBB(P), \leq)$ *is a partial order.*

To establish that the b-bundles of P consistently approximate a largest b-bundle we would further like to obtain that $(IBB(P), \leq)$ is a complete lattice. However, we will now demonstrate that this does not hold. Indeed we have:

Proposition 4.10 $(IBB(P), \leq)$ *is neither a lattice nor a complete partial order.*

Proof. To prove this result we will exhibit two b-bundles that have upper bounds but no least upper bound. The b-bundles (which are also bundles) are presented in Figure 2.

Bundle A contains one instance of trace $+m_1\, +m_2$ and one instance of trace $-m_1\, -m_2$, with the send and receive events matched up in the obvious way. Bundle B contains *two* instances of trace $+m_1\, +m_2$ and *one* instance of trace $-m_1\, -m_2$, with the receive events of the latter matched up to send events of *different* strands. Observe that A and B are incomparable: B can clearly not be *injectively* folded onto A; while there cannot be a homomorphism from A to B because there is no strand in B with two outgoing message-flow arrows.

By a similar argument it is clear that any upper bound of A and B must contain at least two instances of trace $+m_1\, +m_2$ and two instances of trace $-m_1\, -m_2$. If two b-bundles contain the same number of events and are comparable with respect to $\leq$ then there will be a bijective homomorphism between them, and hence an

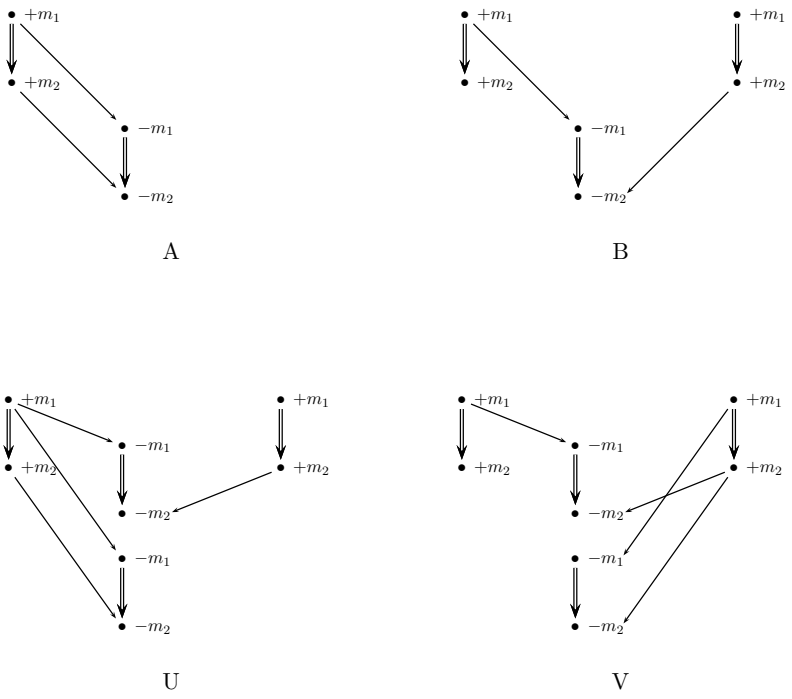


Fig. 2. m_1 and m_2 are ground messages, $+m_1$ stands short for $+$ *fresh* $\emptyset$ *in* m_1, and similarly for $+m_2$. It is clear that there is a protocol P such that A to V are b-bundles of P.

isomorphism. Thus, any upper bound of A and B which contains only eight events is, up to isomorphism, a minimal upper bound of A and B.

Consider b-bundles U and V of Figure 2. It is easy to check that both of them are upper bounds of A and B. Further, both of them have only eight events, and thus, up to isomorphism, they must be minimal upper bounds. On the other hand, U and V are not isomorphic: e.g., U has an event labelled by $+m_1$ with two outgoing message-flow arcs while V does not. □

Since $(IBB(P), \leq)$ is not even a complete partial order, $\leq$ cannot be interpreted as a notion of approximation in the information-ordering sense: a b-bundle that is higher in the order does not extend the information of the elements below in a consistent way. It also indicates that a largest b-bundle might simply not exist. Thus, the theory of branching bundles does not turn out to be very satisfying. Analysing the above counter-example will, however, lead us to a satisfying theory of *symbolic* b-bundles.

Remark 4.11 Those readers who are familiar with the strand space model may wonder whether a counter-example could still be obtained if the $\rightarrow$-relation in bundles was disallowed to be forwards-branching (and the Intruder must duplicate messages explicitly). Note that for b-bundles forwards-branching would still be natural,

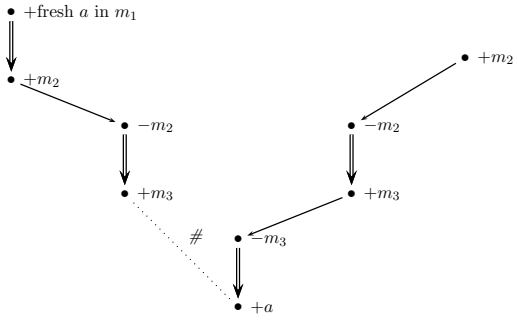


Fig. 3. Assume $a \not\sqsubseteq m_2$ & $a \not\sqsubseteq m_3$.

and a more involved counter-example could be constructed.

5 Symbolic Branching Bundles

Let us take another look at the bundles of Figure 2. One could argue that A and B represent the same information with respect to the Intruder's viewpoint. On the one hand, at both, A and B, the information the Intruder *has gained so far* is essentially the same:

> the input to a strand with trace $-m_1\,-m_2$ can be obtained from the send events of a strand with trace $+m_1\,+m_2$, where instances of the latter trace do not require any input.

On the other hand, the information the Intruder *may gain in the future* is also essentially the same at both, A and B:

> for example, to simulate a future of B by a future of A, if the Intruder employs one of the two $+m_1$-events of B as send input to a future strand, he can use the one $+m_1$-event of A in exactly the same way. Furthermore, if in a future of B each of the two $+m_1\,+m_2$ strands is extended by an action such that the actions are different but non-conflicting, then in A a new strand with trace $+m_1\,+m_2$ can be spawned, so as to obtain two non-conflicting strands with analogous traces.

This is why, on second look, it is not surprising that b-bundles do not form an information ordering: there are many inconsistent ways of representing the same information. On the positive side, this also suggests that we may still obtain an information ordering if we work with a notion of *symbolic* b-bundle.

How could a notion of symbolic b-bundle look like? We would expect that in a symbolic b-bundle all branches that represent essentially the same information are 'glued together', thereby folding a space of b-strands together into a space of *symbolic* b-strands. A space of symbolic b-strands is a b-strand space whose b-strands are considered symbolically. This means we need to relax our notion of conflict. Since a b-strand may now represent several, possibly parallel, sessions, two events that have the same $\Rightarrow$-predecessor are not necessarily in conflict any more:

as long as there is no conflict due to unique origination we could have instantiated parallel sessions.

Definition 5.1 Let $B = (E, \Rightarrow, \rightarrow, GAct, l)$ be a b-strand space of P. Whenever we wish to consider the b-strands of B symbolically then we call B a *symbolic b-strand space*, and redefine the *conflict relation* $\#_B$ by deleting clause (i) from the definition of immediate conflict in Def. 3.2.

Next we need to clarify: when do two branches of a b-bundle represent essentially the same information? One condition is, naturally, that their trace must be the same. However, Figure 3 motivates that we need to adopt a second condition. It illustrates that two strands that have the same trace but different pasts can have different futures: the $+m_3$-event of the strand on the right-hand side can be used as input to the $-m_3$-event of the lowest strand. However, this is not possible for the $+m_3$-event of the strand on the left-hand side, since this would lead to a self-conflict of the $+a$-event due to violation of unique origination. Thus, we only wish to 'glue together' branches when they have the same trace *and* the same past. In more detail: if two branches b and b' have the same trace and for all positions i of the trace, the ith event of b has the same $\rightarrow$-predecessor as the ith event of b' then b and b' are to be identified. We can capture this condition more succinctly in terms of events: if two events have the same label and the same predecessors then they are to be identified. Altogether, this leads to the following definition of symbolic b-bundle.

Definition 5.2 A *symbolic b-bundle* B of P is a symbolic b-strand space such that Axioms (B1) to (B5) and (BB) of the definition of b-bundle hold (the latter with respect to the redefined conflict relation). In addition we impose the following axiom:

SBB for every $e_1, e_2 \in E$, if
(a) $l(e_1) = l(e_2)$, and
(b) $\Rightarrow\text{-}pred(e_1) = \Rightarrow\text{-}pred(e_2)$, and
(c) $\rightarrow\text{-}pred(e_1) = \rightarrow\text{-}pred(e_2)$
then $e_1 = e_2$.

Axiom (SBB) is analogous to, and inspired by, a requirement for Petri net branching processes (Definition 7 of [4]). We also derive a notion of *symbolic bundle*. Naturally, they are based on symbolic b-strands.

Definition 5.3 A *symbolic bundle* is a conflict-free symbolic b-bundle (where the conflict relation is as redefined above).

We will now demonstrate that every b-bundle B of P can indeed be folded onto a symbolic b-bundle. By induction on the causal depth of events we successively identify those events that satisfy conditions (a) to (c) of the definition of symbolic b-bundle. Formally, we define an equivalence relation $\sim$ on $E \cup \{nil\}$ inductively as follows:

(i) $nil \sim nil$,

(ii) $e_1 \sim e_2$ if
  (a) $l(e_1) = l(e_2)$,
  (b) $\Rightarrow\text{-}pred(e_1) \sim \Rightarrow\text{-}pred(e_2)$, and
  (c) $\rightarrow\text{-}pred(e_1) \sim \rightarrow\text{-}pred(e_2)$.

It is easy to check that $\sim$ is indeed an equivalence relation. Denote the equivalence class of event e by $[e]_\sim$. Given a b-bundle B, the *folding of* B, denoted by $fold(B)$, is defined to be the tuple $(E_f, \Rightarrow_f, \rightarrow_f, GAct, l_f)$ where

- $E_f = \{[e]_\sim \mid e \in E\}$,
- $\forall f_1, f_2 \in E_f$, $f_1 \Rightarrow_f f_2$ if and only if $e_1 \Rightarrow_B e_2$ for some $e_1 \in f_1$, $e_2 \in f_2$,
- $\forall f_1, f_2 \in E_f$, $f_1 \rightarrow_f f_2$ if and only if $e_1 \rightarrow_B e_2$ for some $e_1 \in f_1$, $e_2 \in f_2$,
- $\forall f \in E_f$, $l(f) = a$ if and only if $l(e) = a$ for some (or equivalently all) $e \in f$.

Proposition 5.4 *Let B be a b-bundle of P.*

(i) *$fold(B)$ is a symbolic b-bundle of P.*

(ii) *If B is a bundle then $fold(B)$ is a symbolic bundle.*

(iii) *fold is a surjective homomorphism from B to $fold(B)$.*

On the other hand, every symbolic b-bundle can be transformed into a b-bundle by disentangling non-conflicting strands that are glued together. In particular, every symbolic bundle can be transformed into a bundle. (These connections will be formalized in the full version of the paper.)

The transformations give rise to the following observation, which shows how reachability problems on bundles can be reduced to reachability problems on symbolic bundles.

Proposition 5.5 *Assume a protocol P and a finite set of strands S of P. There is a bundle B of P with $S \subseteq strands(B)$ if and only if the events of S are conflict-free and there is a symbolic bundle B' of P with $traces(S) \subseteq traces(B')$. (We use $strands(B)$, $traces(S)$, and $traces(B')$ with the obvious meaning.)*

This shows that it should be adequate to work with symbolic bundles in most situations. Most verification problems for security protocols can be expressed as reachability problems: check whether a situation that represents an attack can be reached. In the strand space model this can be formalized as follows:

Given: A protocol P, and a finite set of strands S.

Decide: Is there a finite bundle B such that $S \subseteq strands(B)$?

6 A Theory of Symbolic Branching Bundles

It is straightforward to carry over all concepts and positive results of Section 4 to symbolic b-bundles. In particular, we have a partial order $\leq$ on $ISBB(P)$, the set of isomorphism classes of symbolic b-bundles of P. However, now we indeed obtain:

Theorem 6.1 *$(ISBB(P), \leq)$ is a complete lattice.*

Due to Axiom (SBB) the theorem can be proved analogously to Engelfriet's

result on Petri net branching processes [4]. The proof will be provided in the full version of the paper.

Theorem 6.1 guarantees the existence of a unique maximal element in $ISBB(P)$, which captures all possible behaviour of P in a symbolic fashion. We call it the *symbolic unfolding* of P. Thus, every protocol has a strand space semantics in terms of its symbolic unfolding. Further, by the results of Section 4.2 this semantics comes with a notion of computation state in terms of symbolic sub-bundles, a transition relation, and close relations to event structures.

It remains to be investigated whether restricting our attention to the symbolic unfolding is indeed suitable in most situations of security protocol analysis. We will also examine whether it can help with the state space explosion problem in model-checking tools. On the theoretical side, the relationship between symbolic b-bundles and b-bundles can be further formalized using category theory.

Acknowledgement: I thank the anonymous referees for their valuable comments.

References

[1] Crazzolara, F. and G. Winskel, *Events in security protocols*, in: *ACM Conference on Computer and Communications Security*, 2001, pp. 96–105.

[2] Crazzolara, F. and G. Winskel, *Petri nets in cryptographic protocols*, in: *IEEE IPDPS-01*, 2001, p. 149.

[3] Crazzolara, F. and G. Winskel, *Composing strand spaces*, in: *FST TCS 2002*, Lecture Notes in Computer Science **2556**, 2002, pp. 97–108.

[4] Engelfriet, J., *Branching processes of Petri nets*, Acta Inf. **28** (1991), pp. 575–591.

[5] Fábrega, F. J. T., J. C. Herzog and J. D. Guttman, *Strand spaces: Why is a security protocol correct?*, in: *Symposium on Security and Privacy* (1998).

[6] Fröschle, S., *The insecurity problem: Tackling unbounded data*, in: *CSF 2007* (2007), pp. 370–384.

[7] MITRE, *The strand space method web page*, http://www.mitre.org/tech/strands/.

Expressiveness Issues in Calculi for Biochemistry

Gianluigi Zavattaro

Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura Anteo Zamboni 7, I-40127 Bologna, Italy.
E-mail: zavattar@cs.unibo.it

Abstract

We explore the computational power of biochemistry with respect to basic chemistry, identifying complexation as the basic mechanism that distinguishes the former from the latter. We use two process algebras, the Chemical Ground Form (CGF) which is equivalent to basic chemistry, and the Biochemical Ground Form (BGF) which is a minimalistic extension of CGF with primitives for complexation. We characterize an expressiveness gap: CGF can only approximate Turing powerful formalisms while BGF supports a finite precise encoding of Random Access Machines, a well-known Turing powerful formalism.

1 Process Calculi for (Bio)Chemistry

In [1] a process calculus, called *Chemical Ground Form* (CGF), was proposed for the compositional description of chemical systems, and proved to be both stochastically and continuously equivalent to chemical kinetics. Chemical kinetics is the traditional model used to represent basic chemical systems in terms of monomolecular and bimolecular reactions between molecules belonging to a predefined set of species. Biochemistry is obviously based on chemistry, and in principle one can always express the behavior of a biochemical system by a collection of chemical reactions. But there is a major practical problem with that approach: the collection of reactions for virtually all biochemical systems is an infinite one. For example, just to express the chemical reactions involved in linear polymerization, we need to have a different chemical species for each length n of polymer P_n, with reactions to grow the polymer: $P_n + M \rightarrow P_{n+1}$. While each polymer is finite, the set of possible polymerization reactions is infinite. Nature adopts a more modular solution: the act of joining two molecules is called *complexation*, and polymers are made by iteratively complexing monomers. Each monomer obeys a *finite* simple set of rules that leads to the formation of polymers of any length. A formalization of basic complexation mechanisms was proposed in [1], where association and dissociation actions model the complexations and decomplexation between two molecules. In [2]

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

a new calculus, called *Biochemical Ground Form* (BGF) was defined simply adding association and dissociation to CGF.

In [2,4] the computational power of basic chemistry and basic biochemistry was studied exploiting the process calculi CGF and BGF as their formalizations, respectively. In [2] an interesting expressiveness gap was proved between CGF and BGF: CGF is not Turing complete while BGF supports a finite precise encoding of Random Access Machines, a well-known Turing powerful formalism. Even if CGF is not Turing complete, in [4] it is shown that Random Access Machines can be at least approximated with an error probability smaller than any arbitrary precision. This interesting result about basic chemistry, discussed in [3], has important consequences as proved in [4]: in basic chemistry the existential termination problem is decidable while universal termination turns out to be undecidable.

References

[1] Cardelli, L., *On Process Rate Semantics*, Theoretical Computer Science **in press** (2008), available at http://dx.doi.org/10.1016/j.tcs.2007.11.012.

[2] Cardelli, L. and G. Zavattaro, *On the Computational Power of Biochemistry*, in: *Proc. of 3rd International Conference on Algebraic Biology (AB'08)*, LNCS **to appear**, 2008.

[3] Soloveichik, D., M. Cook, E. Winfree and J. Bruck, *Computation with Finite Stochastic Chemical Reaction Networks*, Natural Computing **in press** (2008), available at http://dx.doi.org/10.1007/s11047-008-9067-y.

[4] Zavattaro, G. and L. Cardelli, *Termination Problems in Chemical Kinetics*, in: *Proc. of 19th International Conference on Concurrency Theory (CONCUR'08)*, LNCS **to appear**, 2008.

A Calculus for Mobile Ad-hoc Networks with Static Location Binding

Jens Chr. Godskesen [1]

IT University of Copenhagen

Abstract

We present a process calculus for mobile ad hoc networks which is a natural continuation of our earlier work on the process calculus CMAN [6]. Essential to the new calculus is the novel restricted treatment of node mobility imposed by hiding of location names using a *static* binding operator, and we introduce the more general notion of unidirectional links instead of bidirectional links. We define a natural weak reduction semantics and a reduction congruence and prove our weak contextual bisimulation equivalence to be a *sound* and *complete* co-inductive characterization of the reduction congruence.
The two changes to the calculus in [6] yields a much simpler bisimulation semantics, and importantly and in contrast to [6] we manage to provide a *non-contextual* weak bisimulation congruence facilitating ease of proofs and being strictly contained in the contextual bisimulation.

1 Introduction

The widespread use of communicating mobile devices makes mobile and wireless networks become more and more important. The area of application is broad, spanning from ambient intelligence over mobile ad hoc, sensor, and mesh networks, to cellular networks for mobile telephony.

The communication primitive for wireless networks is message broadcast. However in contrast to wired local area networks where broadcasted messages reach every node in the network, for wireless networks broadcast is *local* because messages will only reach the nodes within the communication range of the emitting node. Put differently, in wired networks broadcast scope is *transitive* and *bidirectional* in that if nodes l and m can communicate directly and if m and n can do so also, then in turn l and n can communicate directly whereas this is not necessarily the case for wireless networks.

Our work is devoted to a particular kind of wireless networks, i.e. *Mobile Ad Hoc Networks* (MANETs). MANETs are self organizing without centralized control, and they do not contain a pre-deployed infrastructure for routing messages. A MANET

[1] Supported by grant no. 272-05-0258 from the Danish Research Agency.

[2] Email: jcg@itu.dk

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

may be formed when a collection of nodes join together and agree on how to route messages for each other over possibly multiple hops.

In this paper we present a process calculus for MANETs which is a natural continuation and refinement of our earlier work on the calculus CMAN [6]. Essential to the new calculus is the restricted treatment of node mobility imposed by hiding of location names using a *static* binding operator, this yields a much simpler labelled transition system and bisimulation semantics. To our knowledge no other calculus for MANETs hides nodes and restricts mobility through a calculus operator. Also, we introduce the more general notion of unidirectional links instead of bidirectional links; e.g. because some nodes have larger transmission range than others. We define a natural weak reduction semantics, radically different from the one in [6], and we define a reduction congruence and prove our weak contextual bisimulation equivalence to be a *sound* and *complete* co-inductive characterization of the reduction congruence. Most importantly, and in contrast to [6], we conveniently manage to devise a non-contextual weak bisimulation congruence that is a considerably advantage in many proofs. The non-contextual bisimulation is strictly contained in our reduction congruence.

1.1 Related Work

Despite the widespread use of broadcasting technology it turns out that process calculi for broadcasting systems are not as well-studied as the more common point-to-point calculi like e.g. CCS [9], or in a mobile setting for instance the π-calculus [11,10] and the Ambient Calculus [3]. Moreover, in [4] it is demonstrated that that it is impossible to encode broadcast communication using point-to-point communication uniformly in the π-calculus.

The first study of calculi for broadcasting systems was CBS [15]. Later broadcasting was introduced in a mobile setting in $b\pi$ [5], MBS [16], and HOBS [14]. However, they all let broadcast be transitive and hence are not well suited for local wireless broadcast. More recently local wireless broadcast has been studied in CBS# [13], an extension of CBS. The neighborhood relation between nodes is dealt with letting the semantics be parameterized and quantified over a set of configurations (graphs).

The ω-calculus [17] is an extension of the π-calculus. It is interesting in that the neighborhood relation is modeled by annotating the processes with the *groups* to which they belong. A group is a set of nodes that lie within each others communication range. Local wireless broadcast has also been studied in CMN [7], here the neighborhood relation is taken care of by a metric function that tells if two physical locations are close enough to communicate. [3]

As mentioned above, in [6] we developed CMAN where also the neighborhood relation is explicitly part of the syntax because a node is annotated by the nodes to which it is connected. However, a bit unnaturally, and like in the ω-calculus, communication between nodes is carried out on bidirectional links. Further we assumed, as in CBS#, CMN, and the ω-calculus, that nodes may move and connect

[3] The calculus CWS [8] also studies wireless broadcast but at a much lower level of abstraction, in particular they take the phenomenon of interference into account.

arbitrarily, but to be realistic it is easy to envisage that two particular mobile nodes in a MANET can never meet due to physical obstacles (walls, buildings, etc.). Even though we apply many changes to CMAN in this paper the analysis of a cryptographic routing protocol for MANETs carried out in [6] is still valid for our new calculus.

1.2 Motivation

Our goal in this paper is to develop a process calculus for MANETs where communication links are not assumed to be bidirectional, and moreover we want primitives that restrict the otherwise unrestricted mobility of nodes.

A *node*, $\lfloor p \rfloor_l^\sigma$, in our new calculus is modeled as a *process* p located at some logical *location* (or identity) l and connected to other nodes at locations σ. Nodes composed in parallel constitute a *network*, say

$$\lfloor p \rfloor_l^m \parallel \lfloor q \rfloor_m^n \parallel \lfloor r \rfloor_n \ , \tag{1}$$

where the node at location l, $\lfloor p \rfloor_l^m$, is connected to the node at location m, $\lfloor q \rfloor_m^n$, which in turn is connected to the node at location n, $\lfloor r \rfloor_n$.

Mobility is defined by a simple reduction, say that the node at location n in (1) autonomously moves and becomes *connected* to the node at location l,

$$\lfloor p \rfloor_l^m \parallel \lfloor q \rfloor_m^n \parallel \lfloor r \rfloor_n \searrow \lfloor p \rfloor_l^m \parallel \lfloor q \rfloor_m^n \parallel \lfloor r \rfloor_n^l \ . \tag{2}$$

Dually, nodes may arbitrarily *disconnect*, for instance m disconnects from n in

$$\lfloor p \rfloor_l^m \parallel \lfloor q \rfloor_m^n \parallel \lfloor r \rfloor_n^l \searrow \lfloor p \rfloor_l^m \parallel \lfloor q \rfloor_m \parallel \lfloor r \rfloor_n^l \ . \tag{3}$$

A process $\langle t \rangle.p$ can broadcast t and in so doing become p, and a process $(x).q$ can receive a broadcasted message t becoming $q\{t/x\}$, i.e. q with all free occurrences of x replaced by t. *Local synchronous broadcast* is defined by a network *broadcast reduction* labelled by the location of the node containing the emitting process, say

$$\lfloor \langle n \rangle.p \rfloor_l^{mk} \parallel \lfloor (x).q \rfloor_m \parallel \lfloor (x).r \rfloor_k \searrow_l \lfloor p \rfloor_l^{mk} \parallel \lfloor q\{n/x\} \rfloor_m \parallel \lfloor r\{n/x\} \rfloor_k \ , \tag{4}$$

where $\langle n \rangle.p$ broadcasts n to all nodes to which the node at l is connected.

In CMAN one may choose to hide locations in order to let broadcasting be unobservable, the hiding is carried out by a *scope extensible* binder, νl. For instance, the hidden node $\nu l.\lfloor \langle n \rangle.p \rfloor_l$, may connect to other nodes by first *extruding* its location name (through structural congruence, assuming l is fresh),

$$\nu l.\lfloor \langle n \rangle.p \rfloor_l \parallel \lfloor (x).q \rfloor_m \equiv \nu l.(\lfloor \langle n \rangle.p \rfloor_l \parallel \lfloor (x).q \rfloor_m) \searrow \nu l.(\lfloor \langle n \rangle.p \rfloor_l^m \parallel \lfloor (x).q \rfloor_m) \ ,$$

and subsequently send its messages to its new neighbor,

$$\nu l.(\lfloor \langle n \rangle.p \rfloor_l^m \parallel \lfloor (x).q \rfloor_m) \searrow \nu l.(\lfloor p \rfloor_l^m \parallel \lfloor q\{n/x\} \rfloor_m) \ ,$$

the latter carried out as an *unlabelled* and hence unobservable reduction. As a novelty, in this paper we instead introduce a *static* binder for location names, denoted by $\backslash l$, whose scope cannot be extended and we abolish the scope extensible binder mentioned above. Not only will such a binding outside its scope conceal all broadcasting events carried out at l, but also connectivity involving the node at l

is restricted within the scope of the binder. For instance, in

$$P \parallel \lfloor r \rfloor_k \ , \quad \text{where} \quad P = (\lfloor \langle n \rangle.p \rfloor_l^m \parallel \lfloor (x).q \rfloor_m) \setminus l \tag{5}$$

the location name l is bound and inaccessible to the node at k, so the two nodes cannot connect and hence not directly receive messages from each other. The hidden node in (5) may broadcast to m as demonstrated by

$$P \searrow (\lfloor p \rfloor_l^m \parallel \lfloor q\{n/x\} \rfloor_m) \setminus l \ , \tag{6}$$

but then the broadcast is carried out as an unobservable unlabelled reduction.

1.3 Structure

The paper is organized as follows: Our new calculus is presented in the next section and in Section 3 we define a reduction semantics and a reduction congruence. In Section 4 we define bisimulation equivalences, one being a contextual co-inductive characterization of the reduction congruence, and one being a non-contextual congruence strictly contained in reduction congruence.

We illustrate the application of the calculus on a simple cryptographic message passing protocol where node mobility is restricted, this example could not have been modeled and analyzed in our previous work [6]. Finally, we end by a conclusion.

2 The Calculus

In this section we outline our calculus defining first terms, then processes, and finally networks.

2.1 Terms

Assume an infinite set of *names* $\mathcal{N}$ ranged over by n, an infinite set of *variables* $\mathcal{X}$ ranged over by x, and two disjoint finite sets, $\mathcal{F}$ and $\mathcal{G}$, of *constructor* and *destructor* symbols ranged over by f and g respectively. Then the set of *terms* is defined by the grammar below where f is a constructor symbol with arity k. We let $\mathcal{T}$ denote the set of all terms with no variables.

$$s, t ::= n \mid x \mid f(t_1, \ldots, t_k)$$

2.2 Processes

We assume a set of process variables $\mathcal{Z}$ ranged over by z. The set of *processes* is defined by the grammar

$$p, q ::= 0 \mid \langle t \rangle.p \mid (x).p \mid \mathit{if}\ (t = s)\ \mathit{then}\ p\ \mathit{else}\ q \mid \mathit{let}\, x = t\ \mathit{in}\ p \mid$$

$$\mathit{let}\, x = g(t_1, \ldots, t_i)\ \mathit{in}\ p\ \mathit{else}\ q \mid \nu n.p \mid z \mid \mathit{rec}\ z.p$$

The processes 0, $\nu n.p$, $\mathit{if}\ (t = s)\ \mathit{then}\ p\ \mathit{else}\ q$, $\mathit{let}\ x = t\ \mathit{in}\ p$, and $\mathit{rec}\ z.p$ are standard. [4] The process $\langle t \rangle.p$ may broadcast t and in so doing become p, and $(x).p$

[4] We assume all free occurrences of z in p to be either input or output prefixed.

binds x in p and may receive a term t and replace all free occurrences of x in p by t. Often we write $\langle t \rangle$ for $\langle t \rangle.p$ when p is 0. The process *let* $x = g(t_1, \ldots, t_k)$ *in* p *else* q also binds x in p, if the destructor application $g(t_1, \ldots, t_k)$ evaluates to a term t then x is bound to t in p, otherwise the process becomes q.

We let $p\{t/x\}$ denote p where x is substituted by t. Likewise, $p\{q/z\}$ denotes p where z is substituted by q. The set of *free names* in p is denoted by $fn(p)$, and its *free variables* are denoted by $fv(p)$. A process p is *closed* if $fv(p) = \emptyset$. $\mathbf{P}$ denotes the set of all closed processes and we identify processes up to α-equivalence.

2.3 Networks

Assume a finite set of *location* names $\mathcal{L}$ ranged over by l and k. We assume $\mathcal{N} \cap \mathcal{L} = \emptyset$. We let σ range over sets of location names, and we let ϵ denote the empty set. The set of *networks* is defined by the grammar

$$P, Q ::= 0 \;\mid\; \lfloor p \rfloor_l^\sigma \;\mid\; \nu n.P \;\mid P \setminus \sigma \;\mid\; P \parallel Q$$

The network 0 denotes the empty network. $\lfloor p \rfloor_l^\sigma$ is a node at location l containing the process p and connected to all nodes in σ. $\nu n.P$ is the network P with a new name n bound by νn, $P \setminus \sigma$ denotes a network with locations in σ bound and hidden, and finally $P \parallel Q$ is the parallel composition of the two networks P and Q. We let the new name operator have higher precedence than the hiding operator which in turn has higher precedence than the left associative parallel composition. We write $\lfloor p \rfloor_l$ instead of $\lfloor p \rfloor_l^\epsilon$. When $\tilde{n} = \{n_1, \ldots, n_i\}$ we write $\tilde{n}n$ for $\tilde{n} \cup \{n\}$ and we write $\nu\tilde{n}$ instead of $\nu n_1 \ldots \nu n_i$. We write σl instead of $\sigma \cup \{l\}$, l for $\{l\}$, and $\sigma\sigma'$ for the union of disjoint sets σ and σ'.

The sets of free names, locations, and variables in P, denoted by $fn(P)$, $fl(P)$, and $fv(P)$ respectively, are defined as expected. We let $P\{t/x\}$ denote P where all free occurrences of x in P are substituted by t. We let $P_{l \oplus k}$ denote network P where k is added to the connections at the (free) location l, taking care that k is not bound in P (using α-conversion if needed), formally we define: $(\lfloor p \rfloor_l^\sigma)_{l \oplus k} = \lfloor p \rfloor_l^{\sigma k}$, $(\lfloor p \rfloor_m^\sigma)_{l \oplus k} = \lfloor p \rfloor_m^\sigma$, if $l \neq m$, $(\nu n.P)_{l \oplus k} = \nu n.(P_{l \oplus k})$, $(P \parallel Q)_{l \oplus k} = P_{l \oplus k} \parallel Q_{l \oplus k}$, and $(P \setminus \sigma)_{l \oplus k} = (P_{l \oplus k}) \setminus \sigma$ if $l, k \notin \sigma$. We let $P_{l \oplus \sigma}$ be the obvious generalization of $P_{l \oplus k}$.

We say that a network P is *well-formed* if each node in P is not connected to itself and if each location in P is unique. In the sequel we consider only the set of well-formed networks and we identify networks up to α-equivalence. The set of well-formed and variable closed networks is denoted by $\mathbf{N}$.

3 Reduction Semantics

In this section we provide our calculus with a natural reduction semantics; interestingly and due to the static location binder, the semantics is quite different compared to the one for CMAN.

As in the seminal work on barbed bisimulation [12], and as in [6], we strive to have as simple as possible reduction semantics and to allow an external global observer to have minimal observability, in our case: reductions $\searrow_l$ when the node at the free location l broadcasts, and reductions $\searrow$ for connections, disconnections,

$$let\ x = t\ in\ p\ \equiv_{\mathbf{P}}\ p\{t/x\} \qquad if\ (t = t)\ then\ p\ else\ q \equiv_{\mathbf{P}} p$$

$$if\ (t = s)\ then\ p\ else\ q\ \equiv_{\mathbf{P}}\ q\ ,\quad \text{if } t \neq s \qquad rec\ z.p \equiv_{\mathbf{P}}\ p\{rec\ z.p/z\}$$

$$let\ x = g(t_1, \ldots, t_i)\ in\ p\ else\ q\ \equiv_{\mathbf{P}}\ p\{t/x\}\ ,\quad \text{if } g(t_1, \ldots, t_i) = t$$

$$let\ x = g(t_1, \ldots, t_i)\ in\ p\ else\ q\ \equiv_{\mathbf{P}}\ q\ ,\quad \text{if } g(t_1, \ldots, t_i) \text{ not defined}$$

Table 1
Structural congruence, processes.

$$P \parallel 0 \equiv P \qquad P \parallel Q \equiv Q \parallel P \qquad (P \parallel P') \parallel P'' \equiv P \parallel (P' \parallel P'')$$

$$\lfloor p \rfloor_l^\sigma \equiv \lfloor q \rfloor_l^\sigma\ ,\ \text{if } p \equiv_{\mathbf{P}} q \qquad \lfloor \nu n.p \rfloor_l^\sigma \equiv \nu n.\lfloor p \rfloor_l^\sigma \qquad (\nu n.P) \setminus \sigma \equiv \nu n.(P \setminus \sigma)$$

$$\nu n.0 \equiv 0 \qquad \nu n.\nu n'.P \equiv \nu n'.\nu n.P \qquad \nu n.P \parallel Q \equiv \nu n.(P \parallel Q)\ ,\ \text{if } n \notin fn(Q)$$

Table 2
Structural congruence, networks.

and broadcast from hidden nodes. In particular an observer cannot identify the broadcasted message and the receivers of the message. Indistinguishability under these observations gives rise to a natural equivalence which in turn induces a natural congruence over networks, i.e. the equivalence in all contexts closed under structural congruence.

3.1 Reductions

As usual, a binary relation $\mathcal{R}$ on $\mathbf{P}$ is a *congruence* if $p\ \mathcal{R}\ q$ implies $c(p)\ \mathcal{R}\ c(q)$ for any variable closing process context c. Structural congruence on $\mathbf{P}$, $\equiv_{\mathbf{P}}$, is the least congruence and equivalence relation that is closed under α-conversion and the rules in Table 1. We write $C(P)$ for the insertion of P in the hole of a network context C whenever $C(P)$ is well-formed and variable closed. A relation $\mathcal{R}$ on $\mathbf{N}$ is a *congruence* if $P\ \mathcal{R}\ P'$ implies $C(P)\ \mathcal{R}\ C(Q)$ for all $C(P)$. [5] Structural congruence on $\mathbf{N}$, $\equiv$, is the least congruence and equivalence relation that is closed under α-conversion and the rules in Table 2. The rules are standard except that new names can be extruded from nodes and pass the scope of statically bound location names.

To assist in the definition of the reduction rules we introduce a family of *abstractions* ranged over by A_σ and defined by:

$$A_\epsilon ::= 0 \qquad A_l ::= \lfloor 0 \rfloor_l^\sigma \ \mid\ \lfloor \langle t \rangle.p \rfloor_l^\sigma \ \mid\ \lfloor (x).p \rfloor_l^\sigma \qquad A_{\sigma\sigma'} ::= A_\sigma \parallel A_{\sigma'}$$

$$A_\sigma ::= A_\sigma \parallel P \ \mid\ \nu n.A_\sigma \ \mid\ A_\sigma \setminus \sigma'\ ,\ \text{if } \sigma \cap \sigma' = \emptyset$$

In A_σ all locations in σ are free and hence may receive messages. Given an abstraction A_σ we define $A_\sigma \circ t$, i.e. a network being the application of a term t on locations σ in A_σ, inductively by the rules in Table 3.

We define $\searrow_{l,t} \subseteq \mathbf{N} \times \mathbf{N}$ as the least relation closed under $\equiv$ and satisfying the rules in Table 4. Intuitively, $P \searrow_{l,t} P'$ means that the node at (the free) location l

[5] Notice that any congruence, $\mathcal{R}$, has the property that $P\ \mathcal{R}\ Q$ implies $fl(P) = fl(Q)$ due to the well-formedness criteria.

$$\lfloor 0 \rfloor_l^\sigma \circ t = \lfloor 0 \rfloor_l^\sigma \qquad \lfloor \langle t' \rangle.p \rfloor_l^\sigma \circ t = \lfloor \langle t' \rangle.p \rfloor_l^\sigma \qquad \lfloor (x).p \rfloor_l^\sigma \circ t = \lfloor p\{t/x\} \rfloor_l^\sigma$$

$$(A_\sigma \parallel P) \circ t = (A_\sigma \circ t) \parallel P \qquad (\nu n.A_\sigma) \circ t = \nu n.(A_\sigma \circ t) \ , \text{ if } \ n \notin fn(t)$$

$$(A_\sigma \setminus \sigma') \circ t = (A_\sigma \circ t) \setminus \sigma' \qquad (A_\sigma \parallel A_{\sigma'}) \circ t = (A_\sigma \circ t) \parallel (A_{\sigma'} \circ t)$$

Table 3
Abstraction application.

$$(emp)\ \frac{}{\lfloor \langle t \rangle.p \rfloor_l \searrow_{l,t} \lfloor p \rfloor_l} \qquad (hde_1)\ \frac{P \searrow_{l,t} P'}{P \setminus \sigma \searrow_{l,t} P' \setminus \sigma}\ l \notin \sigma$$

$$(brd)\ \frac{P \searrow_{l,t} P'}{P_{l \oplus \sigma} \parallel A_\sigma \searrow_{l,t} P'_{l \oplus \sigma} \parallel (A_\sigma \circ t)} \qquad (cls)\ \frac{P \searrow_{l,t} P'}{P \searrow_l P'}$$

$$(hde_2)\ \frac{P \searrow_l P'}{P \setminus \sigma \searrow_l P' \setminus \sigma}\ l \notin \sigma \qquad (hde_3)\ \frac{P \searrow_l P'}{P \setminus \sigma \searrow P' \setminus \sigma}\ l \in \sigma$$

$$(con)\ \frac{}{\lfloor p \rfloor_l^\sigma \searrow \lfloor p \rfloor_l^{\sigma k}} \qquad (dis)\ \frac{}{\lfloor p \rfloor_l^{\sigma k} \searrow \lfloor p \rfloor_l^\sigma}$$

Table 4
Reduction rules.

has completed broadcasting t to all nodes to which it is connected. A reduction due to rule (emp) describes that a node may broadcast to the empty set of receivers, whereas rule (brd) allows auxiliary nodes σ to be connected to a node l and let the nodes in σ synchronously receive t, whenever l has otherwise completed its broadcast of t. As an example, since $\lfloor \langle n \rangle.p \rfloor_l \searrow_{l,n} \lfloor p \rfloor_l$ we obtain

$$\lfloor \langle n \rangle.p \rfloor_l^m \parallel \lfloor (x).q \rfloor_m \searrow_{l,n} \lfloor p \rfloor_l^m \parallel \lfloor q\{n/x\} \rfloor_m \ , \tag{7}$$

from (brd), and from (7) and rule (brd) we get

$$\lfloor \langle n \rangle.p \rfloor_l^{mk} \parallel \lfloor (x).q \rfloor_m \parallel \lfloor (x).r \rfloor_k \searrow_{l,n} \lfloor p \rfloor_l^{mk} \parallel \lfloor q\{n/x\} \rfloor_m \parallel \lfloor r\{n/x\} \rfloor_k \ . \tag{8}$$

Rule (hde_1) in Table 4 allows free locations to broadcast a term.

We define $\searrow_l \subseteq \mathbf{N} \times \mathbf{N}$ as the least relation closed under $\equiv$, new name, parallel composition, and satisfying the rules in Table 4. Intuitively, $P \searrow_l P'$ means that the node at location l has completed broadcasting some message as indicated by rule (cls). Rule (hde_2) allows broadcast from free locations. As an example, the reduction (4) in the Introduction is inferred from (8) and rule (cls), and from (4) we may further infer

$$\nu n.\lfloor \langle n \rangle.p \rfloor_l^{mk} \parallel \lfloor (x).q \rfloor_m \parallel \lfloor (x).r \rfloor_k \searrow_l \nu n.(\lfloor p \rfloor_l^{mk} \parallel \lfloor q\{n/x\} \rfloor_m \parallel \lfloor r\{n/x\} \rfloor_k) \ ,$$

which does not belong to the reductions in $\searrow_{l,n}$.

Finally, we define $\searrow \subseteq \mathbf{N} \times \mathbf{N}$ as the least relation closed under $\equiv$, new name, parallel composition, and location hiding, and satisfying the rules in Table 4. $P \searrow P'$ is either the result of a hidden broadcast, i.e. rule (hde_3), or a connection or disconnection as defined by the rules (con) and (dis) respectively. For instance, the reduction (6) in the Introduction is inferred from (7) and rule (hde_3), and (2) and (3) are inferred from (con) and (dis) respectively.

3.2 Reduction Congruence

Next we introduce a natural weak congruence in which reductions $\searrow_l$ are our only observables. Let $\searrow^*$ be the reflexive and transitive closure of $\searrow$. We say that a binary relation $\mathcal{R}$ on $\mathbf{N}$ is *weakly reduction-closed* if whenever $P\ \mathcal{R}\ Q$ then $P \searrow_l P'$ ($P \searrow P'$) implies the existence of some Q' such that $Q \searrow^* \searrow_l \searrow^* Q'$ ($Q \searrow^* Q'$) and $P'\ \mathcal{R}\ Q'$.

Definition 3.1 A symmetric relation $\mathcal{R}$ on $\mathbf{N}$ is a *weak reduction congruence* if it is weakly reduction-closed and a congruence.

Let $\cong$ be the largest weak reduction congruence. As an example, $P \cong Q$ if $fl(P) = fl(Q)$ and if neither P nor Q can ever broadcast since no context can distinguish them apart, in particular $0 \cong P$ if $fl(P) = \emptyset$.

4 Bisimulation Semantics

In this section we first provide a labelled transition system; interestingly the network semantics turns out much simpler than the one for CMAN. Next, we give the definition of a weak bisimulation, $\approx$, a sound and complete co-inductive characterization of $\cong$. Also this definition is quite different from the corresponding weak bisimulation for CMAN, but it is still contextual. Therefore, as a novelty compared to [6], we define a non-contextual weak bisimulation that is strictly contained in $\approx$, and we demonstrate its convenience in our examples.

4.1 Labeled Transition System Semantics

We begin with the process semantics and continue with semantics for networks.

4.1.1 Process Semantics

$$(out)\ \frac{}{\langle t\rangle.p \xrightarrow{\langle t\rangle} p} \qquad (open)\ \frac{p \xrightarrow{\nu\tilde{n}\langle t\rangle} p'}{\nu n.p \xrightarrow{\nu\tilde{n}n\langle t\rangle} p'}\ n \in fn(t) \setminus \tilde{n}$$

$$(in_1)\ \frac{}{(x).p \xrightarrow{(t)} p\{t/x\}} \qquad (in_2)\ \frac{}{\langle t'\rangle.p \xrightarrow{(t)} \langle t'\rangle.p} \qquad (in_3)\ \frac{}{0 \xrightarrow{(t)} 0}$$

$$(new)\ \frac{p \xrightarrow{\lambda} p'}{\nu n.p \xrightarrow{\lambda} \nu n.p'}\ n \notin fn(\lambda) \cup bn(\lambda)$$

Table 5
Transition Rules, Processes.

Let the set of *process actions*, $\mathcal{A}_{\mathbf{P}}$, where $t \in \mathcal{T}$, be defined by:

$$\lambda ::= (t) \mid \nu\tilde{n}\langle t\rangle$$

The action (t) describes that t is received by a process and the action $\nu\tilde{n}\langle t\rangle$ denotes the emission of the term t with names in $\tilde{n}$ bound. If $\tilde{n} = \emptyset$ we write $\langle t\rangle$ instead of $\nu\emptyset\langle t\rangle$. We let $fn(\lambda)$ ($bn(\lambda)$) denote the free (bound) names in λ.

The processes semantics is defined by $(\mathbf{P}, \mathcal{A}_{\mathbf{P}}, \rightarrow)$ where $\rightarrow \subseteq \mathbf{P} \times \mathcal{A}_{\mathbf{P}} \times \mathbf{P}$ is the least relation defined by the rules in Table 5 and closed by $\equiv_{\mathbf{P}}$. The rules (out) and (in_1) are immediate, and (in_2) and (in_3) state that processes may lose messages. The rule (new) is standard and the rule $(open)$ takes care of extrusion of new names.

4.1.2 Networks Semantics

The set of *network actions* $\mathcal{A}$ ranged over by α is defined by:

$$\alpha ::= \bar{l}\sigma\nu\tilde{n}\langle t\rangle \mid \sigma(t) \mid \beta \qquad \beta ::= \bar{l} \mid \tau$$

where $t \in \mathcal{T}$. The action $\bar{l}\sigma\nu\tilde{n}\langle t\rangle$ means that the node at location l broadcasts t to nodes in σ where the names in $\tilde{n}$ are bound. $\sigma(t)$ means that t is received by the nodes in σ. $\bar{l}$ denotes that the broadcast session for the node at l has completed. As usual τ denotes an internal computation. We let $bn(\alpha)$ ($fn(\alpha)$) denote the bound (free) names in α, and we let $fl(\alpha)$ denote the free locations in α.

$$(brd)\frac{p \xrightarrow{\nu\tilde{n}\langle t\rangle} p'}{\lfloor p\rfloor_l^{\sigma} \xrightarrow{\bar{l}\sigma\nu\tilde{n}\langle t\rangle} \lfloor p'\rfloor_l^{\sigma}} \qquad (rec_1)\frac{p \xrightarrow{(t)} p'}{\lfloor p\rfloor_l^{\sigma} \xrightarrow{l(t)} \lfloor p'\rfloor_l^{\sigma}} \qquad (rec_2)\frac{}{P \xrightarrow{\epsilon(t)} P}$$

$$(rec_3)\frac{P \xrightarrow{\sigma(t)} P' \quad Q \xrightarrow{\sigma'(t)} Q'}{P \parallel Q \xrightarrow{\sigma\sigma'(t)} P' \parallel Q'} \qquad (opn)\frac{P \xrightarrow{\bar{l}\sigma\nu\tilde{n}\langle t\rangle} P'}{\nu n.P \xrightarrow{\bar{l}\sigma\nu\tilde{n}n\langle t\rangle} P'} \quad n \in fn(t) \setminus \tilde{n}$$

$$(syn)\frac{P \xrightarrow{\bar{l}\sigma\sigma'\nu\tilde{n}\langle t\rangle} P' \qquad Q \xrightarrow{\sigma'(t)} Q'}{P \parallel Q \xrightarrow{\bar{l}\sigma\nu\tilde{n}\langle t\rangle} P' \parallel Q'} \quad \tilde{n} \cap fn(Q) = \sigma \cap fl(Q) = \emptyset$$

$$(cls)\frac{P \xrightarrow{\bar{l}\epsilon\nu\tilde{n}\langle t\rangle} P'}{P \xrightarrow{\bar{l}} \nu\tilde{n}.P'} \qquad (new)\frac{P \xrightarrow{\alpha} P'}{\nu n.P \xrightarrow{\alpha} \nu n.P'} \quad n \notin fn(\alpha) \cup bn(\alpha)$$

$$(hde_1)\frac{P \xrightarrow{\alpha} P'}{P \setminus \sigma \xrightarrow{\alpha} P' \setminus \sigma} \quad fl(\alpha) \cap \sigma = \emptyset \qquad (hde_2)\frac{P \xrightarrow{\bar{l}} P'}{P \setminus \sigma \xrightarrow{\tau} P' \setminus \sigma} \quad l \in \sigma$$

$$(con)\frac{}{\lfloor p\rfloor_l^{\sigma} \xrightarrow{\tau} \lfloor p\rfloor_l^{\sigma k}} \qquad (dis)\frac{}{\lfloor p\rfloor_l^{\sigma k} \xrightarrow{\tau} \lfloor p\rfloor_l^{\sigma}}$$

$$(par)\frac{P \xrightarrow{\beta} P'}{P \parallel Q \xrightarrow{\beta} P' \parallel Q} \quad fl(\beta) \cap fl(Q) = \emptyset$$

Table 6
Transition Rules, Networks.

The semantics for networks is defined by $(\mathbf{N}, \mathcal{A}, \rightarrow)$ where $\rightarrow \subseteq \mathbf{N} \times \mathcal{A} \times \mathbf{N}$ is the least relation satisfying the rules in Table 6, omitting the symmetric counter parts of rules (syn) and (par). The rules (new), (hde_1), and (par) are as expected. The rule (con) deals with connectivity, and so does (dis). As an example, consider the network

$$P = \nu n.(Q \setminus m) \parallel \lfloor (x).p\rfloor_k \ , \quad Q = \lfloor \langle n\rangle.q\rfloor_l \parallel R \ , \quad R = \lfloor (x).r\rfloor_m \parallel \lfloor (x).r'\rfloor_{m'} \ .$$

Using rules (con), (par), (hde_1), and (new) we may get

$$P \xrightarrow{\tau} \nu n.(Q_{l\oplus k} \setminus m) \parallel \lfloor (x).p\rfloor_k = P_{l\oplus k}$$

The rule (brd) states that a node may broadcast to all those nodes to which it is currently connected, (rec_1) defines when a single node can receive a message, and (rec_3) defines when multiple nodes can receive a message. Not all nodes in a parallel compostion are required to receive because of (rec_2), for instance $R \xrightarrow{m'(t)} \lfloor (x).r \rfloor_m \parallel \lfloor r'\{t/x\} \rfloor_{m'}$. The rule ($syn$) defines synchronization of broadcasting enforcing no name clash. For instance, assuming $n \notin fn(r) \cup fn(r')$,

$$Q_{l \oplus \{m,m'\}} \xrightarrow{\bar{l}\epsilon\langle n \rangle} \lfloor q \rfloor_l^{mm'} \parallel (\lfloor r\{n/x\} \rfloor_m \parallel \lfloor r'\{n/x\} \rfloor_{m'}) \ ,$$

so due to (cls), which closes a broadcast session, we get

$$Q_{l \oplus \{m,m'\}} \xrightarrow{\bar{l}} \lfloor q \rfloor_l^{mm'} \parallel (\lfloor r\{n/x\} \rfloor_m \parallel \lfloor r'\{n/x\} \rfloor_{m'}) \ .$$

Observe that $Q_{l \oplus m} \not\xrightarrow{\bar{l}m\langle n \rangle}$ because $m \in fl(R)$, and notice also that rule (rec_2) will

$$C^{\epsilon}_{l,\epsilon} \circ t = C^{\epsilon}_{l,\epsilon} \qquad C_{l,\sigma}((-) \setminus \sigma') \circ t = (C_{l,\sigma} \circ t)((-) \setminus \sigma')$$
$$C_{l,\sigma}((-) \parallel A_{\sigma'}) \circ t = (C_{l,\sigma} \circ t)((-) \parallel A_{\sigma' \circ t})$$

Table 7
Network context application, $C_{l,\sigma}$.

allow locations m and m' to be bypassed in $Q \xrightarrow{\bar{l}} \lfloor q \rfloor_l \parallel R$. The rule ($hde_2$) conceals the emitter of the broadcasted message, so e.g.

$$Q_{l \oplus \{m,m'\}} \setminus l \xrightarrow{\tau} (\lfloor q \rfloor_l^{m,m'} \parallel (\lfloor r\{n/x\} \rfloor_m \parallel \lfloor r'\{n/x\} \rfloor_{m'})) \setminus l \ .$$

Observe that $Q_{l \oplus k} \setminus k \not\xrightarrow{\bar{l}k\langle n \rangle}$ because of (hde_2). The rule (opn) takes care of extrusion of bound (term) names, hence

$$P_{l \oplus k} \xrightarrow{\bar{l}} \nu n.((\lfloor q \rfloor_l^k \parallel R) \setminus m \parallel \lfloor p\{n/x\} \rfloor_k) \ .$$

4.1.3 Correspondence

The correspondence between the transition semantics and structural equivalence is demonstrated by the lemma below.

Lemma 4.1 *If* $P \xrightarrow{\alpha} P'$ *and* $P \equiv Q$ *then there exists* Q' *such that* $Q \xrightarrow{\alpha} Q'$ *and* $P' \equiv Q'$.

and the correspondence between the transition and the reduction semantics is demonstrated by Lemma 4.2 and 4.3.

Lemma 4.2 $P \xrightarrow{\bar{l}} \equiv P'$ *iff* $P \searrow_l P'$.

Lemma 4.3 $P \xrightarrow{\tau} \equiv P'$ *iff* $P \searrow P'$.

4.2 Weak Contextual Bisimulation

Based on the network semantics given in the preceding section below we define our weak contextual bisimulation. First we introduce a subset of network contexts ranged over by $C^{\sigma'}_{l,\sigma}$ and defined by the grammar

$$C^{\epsilon}_{l,\epsilon} ::= (-)$$

$$C^{\sigma''}_{l,\sigma\sigma'} ::= C^{\sigma''}_{l,\sigma}((-) \parallel A_{\sigma'}) \ , \quad \text{if } l \notin fl(A_{\sigma'})$$

$$C^{\sigma'\sigma''}_{l,\sigma} ::= C^{\sigma'}_{l,\sigma}((-) \setminus \sigma'') \ , \quad \text{if } \sigma l \cap \sigma'' = \epsilon$$

Notice that σ' binds free locations of P in $C^{\sigma'}_{l,\sigma}(P)$. We write $C_{l,\sigma}$ instead of $C^{\sigma'}_{l,\sigma}$ if σ' is not important. Given $C_{l,\sigma}$ we write $C_{l,\sigma} \circ t$ for the network context being the application of t on all locations σ in $C_{l,\sigma}$ as defined in Table 7. We write $C_{l,\sigma} \circ (\tilde{n}, t, P)$ for $\nu\tilde{n}.((C_{l,\sigma} \circ t)(P))$ assuming that $\tilde{n}$ does not overlap with the free names in $C_{l,\sigma}$.

Intuitively, for all $C_{l,\sigma}(P)$, if $l \in fl(P)$ then the node at location l in P may broadcast messages to all nodes in $C_{l,\sigma}$ with locations in σ as demonstrated by the Lemma below:

Lemma 4.4 *For all $C_{l,\sigma}(P)$, if $P \xrightarrow{\bar{l}\sigma\nu\tilde{n}\langle t\rangle} P'$ then $C_{l,\sigma}(P) \xrightarrow{\bar{l}} C_{l,\sigma} \circ (\tilde{n}, t, P')$.*

$$\begin{array}{l}
(-) \parallel C_{l,\sigma'}(\lfloor\langle t\rangle.p\rfloor^{\sigma\sigma'}_{l}) \circ t = (-) \parallel (C_{l,\sigma'} \circ t)(\lfloor p\rfloor^{\sigma\sigma'}_{l}) \\
\langle t\rangle D_{l,\sigma\sigma'}((-) \parallel A_{\sigma'}) \circ t = (D_{l,\sigma\sigma'} \circ t)((-) \parallel (A_{\sigma'} \circ t)) \\
\langle t\rangle D_{l,\sigma}((-) \setminus \sigma') \circ t = (D_{l,\sigma} \circ t)((-) \setminus \sigma')
\end{array}$$

Table 8
Network context application, $\langle t\rangle D_{l,\sigma}$.

Also we define a set of network contexts ranged over by $\langle t\rangle D_{l,\sigma}$ and defined by the grammar:

$$\langle t\rangle D_{l,\sigma} ::= (-) \parallel C^{\sigma''}_{l,\sigma'}(\lfloor\langle t\rangle.p\rfloor^{\sigma\sigma'}_{l}) \ , \text{ if } \sigma \cap (\sigma'' \cup fl(C^{\sigma''}_{l,\sigma'})) = \epsilon$$

$$\langle t\rangle D_{l,\sigma} ::= \langle t\rangle D_{l,\sigma\sigma'}((-) \parallel A_{\sigma'})$$

$$\langle t\rangle D_{l,\sigma} ::= \langle t\rangle D_{l,\sigma}((-) \setminus \sigma') \ , \text{ if } \sigma l \cap \sigma' = \epsilon$$

Moreover, for a context $\langle t\rangle D_{l,\sigma}$ we write $D_{l,\sigma} \circ t$ for the context defined by the rules in Table 8. To clarify, for any $\langle t\rangle D_{l,\sigma}(P)$ if $\sigma \subseteq fl(P)$ then all nodes at locations σ in P may receive t broadcasted by the node at location l in $\langle t\rangle D_{l,\sigma}$ as illustrated by:

Lemma 4.5 *For all $\langle t\rangle D_{l,\sigma}(P)$, if $P \xrightarrow{\sigma(t)} P'$ then $\langle t\rangle D_{l,\sigma}(P) \xrightarrow{\bar{l}} (D_{l,\sigma} \circ t)(P')$.*

4.2.1 Weak Contextual Bisimulation

Making use of the two types of contexts outlined above we next define *weak contextual bisimulation*. As usual we let $\stackrel{\tau}{\Longrightarrow}$ be the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$ and we define $\stackrel{\bar{l}}{\Longrightarrow}$ by $\stackrel{\tau}{\Longrightarrow}\stackrel{\bar{l}}{\longrightarrow}\stackrel{\tau}{\Longrightarrow}$.

Definition 4.6 A symmetric relation $\mathcal{R}$ on $\mathbf{N}$ is a *weak contextual bisimulation* if $P\ \mathcal{R}\ Q$ implies

$$\textit{if } P \stackrel{\tau}{\longrightarrow} P' \textit{ then } \exists Q'.\ Q \stackrel{\tau}{\Longrightarrow} Q' \textit{ and } P'\ \mathcal{R}\ Q'$$

$$\textit{if } P \stackrel{\bar{l}\sigma\nu\tilde{n}\langle t\rangle}{\longrightarrow} P' \textit{ then } \forall C_{l,\sigma}(Q).\ \exists Q'.\ C_{l,\sigma}(Q) \stackrel{\bar{l}}{\Longrightarrow} Q' \textit{ and } C_{l,\sigma} \circ (\tilde{n}, t, P')\ \mathcal{R}\ Q'$$

$$\textit{if } P \stackrel{\sigma(t)}{\longrightarrow} P' \textit{ then } \forall \langle t\rangle D_{l,\sigma}(Q).\ \exists Q'.\langle t\rangle D_{l,\sigma}(Q) \stackrel{\bar{l}}{\Longrightarrow} Q' \textit{ and } (D_{l,\sigma} \circ t)(P')\ \mathcal{R}\ Q'$$

We let $\approx$ denote the largest weak contextual bisimulaiton.

Theorem 4.7 *$\approx$ is an equivalence relation and a congruence.*

Example 4.8 It is obvious that $\lfloor\langle t\rangle.\langle s\rangle\rfloor_l \not\approx \lfloor\langle s\rangle.\langle t\rangle\rfloor_l$ if $t \neq s$. However, similar to what is shown in [7] the order of infinite broadcast sequences may be interchanged, i.e. whenever C binds l then

$$C(\lfloor rec\ z.\langle t\rangle.\langle s\rangle.z\rfloor_l) \approx C(\lfloor rec\ z.\langle s\rangle.\langle t\rangle.z\rfloor_l) \tag{9}$$

Intuitively, the reason why (9) holds is that receivers may disconnect from l before a term is broadcasted and connect again in order to receive next.

The first clause in Definition 4.6 is standard. The second clause says that whenever node l in P is able to broadcast to nodes σ in the environment, then when Q is placed in any such environment l in Q must complete a broadcast, but we do not know the receiving nodes. Dually, the third clause states that whenever nodes σ in P synchronously may receive a broadcasted message from the environment then when Q is placed in any such environment the emitting node must complete a broadcast, but again we may not know the actual receiving nodes. The giving up of knowing the broadcast receivers in the matching part of the two latter clauses in Definition 4.6 is related to the fact that in the observables of our reduction semantics we only know the broadcasting node, but we have no means of telling which nodes actually received the broadcasted message.

A major and non-trivial result of this paper is that weak bisimulation is a sound and complete characterization of reduction congruence.

Theorem 4.9 *$\approx = \cong$.*

4.3 Weak Non-Contextual Bisimulation

Because weak contextual bisimulation uses quantification over all contexts it may be hard to show equivalence between two networks, hence we provide a standard non-contextual weak bisimulation letting γ be a network action defined by the grammar:

$$\gamma ::= \bar{l}\sigma\nu\tilde{n}\langle t\rangle \ \mid\ \sigma(t) \ \mid\ \tau$$

Definition 4.10 A symmetric relation $\mathcal{R}$ on $\mathbf{N}$ is a *weak bisimulation* if $P\ \mathcal{R}\ Q$ implies

$$\textit{if } P \xrightarrow{\gamma} P' \textit{ then } \exists Q'.\, Q \stackrel{\gamma}{\Longrightarrow} Q' \textit{ and } P'\ \mathcal{R}\ Q'$$

The largest weak bisimulation, $\dot{\approx}$, is an equivalence relation and a congruence, and

Theorem 4.11 $\dot{\approx} \subset \approx$.

Notice that in contrast to weak contextual bisimulation in a weak bisimulation a matching network must output exactly the same term and also let exactly the same nodes synchronously receive a term. For instance, if f and g are two unary constructors with no destructors then $\lfloor \nu n.\nu m.\langle n\rangle.\langle m\rangle \rfloor_l$ and $\lfloor \nu n.\langle g(n)\rangle.\langle f(n)\rangle \rfloor_l$ are weak contextual bisimilar because for both two unrelated values are broadcasted that are different from any value any context can build, but clearly the two nodes are not weak bisimilar.

Weak bisimulation abstracts from connectivity in that $P_{l\oplus k} \dot{\approx}\ P$ because $P_{l\oplus k} \stackrel{\tau}{\Longrightarrow} P$ and $P \stackrel{\tau}{\Longrightarrow} P_{l\oplus k}$. This property is a characteristic of MANETs in that connection to any reachable node may be obtained and also it turns out useful in many proofs. The adequacy of weak bisimulation is further illustrated by Example 4.12 below which would have been quite hard to show in case of just weak contextual bisimulation. The example illustrates the use of the new feature with restricted mobility and could not have been modelled by the calculus in [6].

$$p \stackrel{\text{def}}{=} \nu n.\langle enc(pair(msg, n), key)\rangle.p' \qquad r \stackrel{\text{def}}{=} (x).\langle x\rangle.r$$
$$p' \stackrel{\text{def}}{=} (x).let\ x' = dec(x, key)\ in\ if\ (x' = n)\ then\ p\ else\ p'\ else\ p'$$
$$q \stackrel{\text{def}}{=} (x).let\ x' = dec(x, key)\ in\ let\ x'' = snd(x')\ in\ \langle enc(x'', key)\rangle.q\ else\ q$$

Table 9
A simple cryptographic message passing protocol.

Example 4.12 Suppose a node, $\lfloor p \rfloor_{l_0}$, that repetitively sends a message, *msg*, to a node, $\lfloor q \rfloor_{l_1}$. The message *msg* is re-broadcasted by p only when the reception of the previous *msg* has been acknowledged. A simple example with only one intermediary node, $\lfloor r \rfloor_{l_2}$, that can communicate with both l_0 and l_1, and where l_0 and l_1 are outside reach of each other, so they must communicate via l_2, may be defined by:

$$P = \nu\, key.(\lfloor p \rfloor_{l_0} \parallel (\lfloor r \rfloor_{l_2} \parallel \lfloor q \rfloor_{l_1}) \setminus l_1) \setminus l_2$$

where *key* is a secret symmetric key shared between p and q. Notice that only q can return the encrypted acknowledge expected by p. Further, let $pair(x, y)$ be a constructor for pairs and let *snd* be the destructor returning the second element of a pair. Also, let $enc(x, y)$ be a constructor denoting the symmetric encryption of a message x by the key y and let *dec* be the corresponding decryption destructor defined by: $dec(enc(x, y), y) = x$. We define p, q, and r in Table 9 using equations instead of recursion. Despite the risk of having a copy of *msg* forward broadcasted by each of two intermediary nodes one may show that one or two intermediary nodes

will not make any observational difference, i.e. $P \dot{\approx} Q$ where

$$Q = \nu\, key.(\lfloor p \rfloor_{l_0} \parallel (\lfloor r \rfloor_{l_2} \parallel \lfloor r \rfloor_{l_3} \parallel \lfloor q \rfloor_{l_1}) \setminus l_1) \setminus \{l_2, l_3\}$$

5 Conclusion

The main contribution of this paper is the refinement of CMAN [6] to allow for restricted node mobility through the novel introduction of a *static location binder*, and also we imposed the more realistic use of *unidirectional* instead of bidirectional links. Importantly the refinement gives rise to a much simpler labelled transition system and bisimulation semantics than in [6]. Moreover, we have developed a natural reduction semantics and congruence, $\cong$, for which the largest weak contextual bisimulation, $\approx$, is a co-inductive sound and complete characterization. Most significantly and in contrast to [6] we manage to define a *non-contextual* weak bisimulation where the largest bisimulation, $\dot{\approx}$, is strictly contained in $\approx$ and which turned out adequate in the proofs of our examples.

Several further developments of our calculus are immediate: For instance the process language could easily be extended with concurrency, and one may consider extending the language with *active substitutions* as in [1] in order to have a less contextual characterization of $\cong$. Moreover, instead of just restricting mobility of nodes we could enforce explicit mobility models as described in [2]. Also, we plan to investigate other equivalences, in particular we want to consider equivalences where the observer is mobile and has only a limited and not a global view of the whole network, and we want to investigate equivalences suitable to help reason about MANETs, and in particular routing and secure routing.

References

[1] Abadi, M. and C. Fournet, *Mobile vales, new names, and secure communication*, in: H. R. Nielson, editor, *28th ACM Symposium on Principles of Programming Languages* (2001), pp. 104–115.

[2] Camp, T., J. Boleng and V. Davies, *A survey of mobility models for ad hoc network research*, Wireless Comm. & Mobile Comp.: Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications **2** (2002), pp. 483–502.
URL citeseer.ist.psu.edu/camp02survey.html

[3] Cardelli, L. and A. Gordon, *Mobile ambients*, in: *Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98* (1998).
URL citeseer.ist.psu.edu/cardelli98mobile.html

[4] Ene, C. and T. Muntean, *Expressiveness of point-to-point versus broadcast communications*, in: *Fundamentals of Comp. Theory*, 1999, pp. 258–268.
URL citeseer.ist.psu.edu/ene99expressiveness.html

[5] Ene, C. and T. Muntean, *A broadcast-based calculus for communicating systems*, in: *6th International Workshop on formal Methods for Parallel Programming: Theory and Applications*, San Francisco, 2001.

[6] Godskesen, J., *A calculus for mobile ad hoc networks*, in: *Proceedings of the 9th International Conference, COORDINATION 2007*, LNCS **4467** (2007), pp. 132–150.

[7] Merro, M., *An observational theory for mobile ad hoc networks*, Electron. Notes Theor. Comput. Sci. **173** (2007), pp. 275–293.

[8] Mezzetti, N. and D. Sangiorgi, *Towards a calculus for wireless systems.*, Electr. Notes Theor. Comput. Sci. **158** (2006), pp. 331–353.

[9] Milner, R., "Communication and Concurrency," Series in Computer Science, Prentice–Hall International, 1989.

[10] Milner, R., "Communicating and Mobile Systems: the π-Calculus," Cambridge University Press, 1999.

[11] Milner, R., J. Parrow and D. Walker, *A calculus of mobile processes, part I/II*, Journal of Information and Computation **100** (1992), pp. 1–77.

[12] Milner, R. and D. Sangiorgi, *Barbed bisimulation*, in: *Proceedings ICALP '92*, Lecture Notes in Computer Science **623** (1992), pp. 685–695.
URL citeseer.ist.psu.edu/milner92barbed.html

[13] Nanz, S. and C. Hankin, *A framework for security analysis of mobile wireless networks*, Theoretical Computer Science **367** (2006), pp. 203–227.

[14] Ostrovsky, K., K. V. S. Prasad and W. Taha, *Towards a primitive higher order calculus of broadcasting systems*, in: *PPDP '02: Proceedings of the 4th ACM SIGPLAN international conference on Principles and practice of declarative programming* (2002), pp. 2–13.

[15] Prasad, K. V. S., *A calculus of broadcasting systems*, Sci. Comput. Program. **25** (1995), pp. 285–327.

[16] Prasad, K. V. S., *A prospectus for mobile broadcasting systems*, Electr. Notes Theor. Comput. Sci. **162** (2006), pp. 295–300.

[17] Singh, A., C. Ramakrishnan and S. Smolka, *A process calculus for mobile ad hoc networks*, www.lmc.cs.sunysb.edu/~cram/Papers/SRS_OmegaCalc2006/.

Labeled transitions for mobile ambients (as synthesized via a graphical encoding)

Filippo Bonchi, Fabio Gadducci, Giacoma Valentina Monreale

Abstract

The paper presents a case study on the synthesis of labeled transition systems (LTSs) for process calculi, choosing as testbed Cardelli and Gordon's Mobile Ambients (MAs). The proposal is based on a graphical encoding: each process is mapped into a graph equipped with suitable interfaces, such that the denotation is fully abstract with respect to the usual structural congruence. Graphs with interfaces are amenable to the synthesis mechanism proposed by Ehrig and König and based on borrowed contexts (BCs), an instance of relative pushouts, introduced by Leifer and Milner. The BC mechanism allows the effective construction of a LTS that has graphs with interfaces as both states and labels, and such that the associated bisimilarity is automatically a congruence. Our paper focuses on the analysis of a LTS over (processes as) graphs with interfaces, as distilled by exploiting the graphical encoding of MAs. In particular, we use the LTS on graphs to recover a suitable LTS directly defined over the structure of MAs processes.

1 Introduction

Among recently introduced process calculi, mobile ambients [8] (MAs) possibly represents the most fruitful proposal so far. The analogy between ambients and network domains, explicitly addressed since the beginning, and between ambients and molecular environments, often exploited in system biology [23], made MAs a centerpiece in recent applications and development of the process calculi paradigm.

It is then baffling that the calculus has been so resilient to the introduction of an observational semantics, based on a labeled transition system (LTS). Indeed, after Milner's treatment of π-calculus [20], it is now customary to present the semantics of a calculus with a reduction semantics, modulo a congruence equating those processes which intuitively represent the same distributed system. As for the case of MAs, the set of rules defining the original reduction semantics is rather complex. Indeed, the system evolution stating the exporting of a process P out of an ambient named n is represented by the rule

$$m[n[out\ m.P|Q]|R] \to n[P|Q]|m[R]$$

[1] Research partially supported by the IST 2004-16004 SEnSOria.

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

The rule needs to carry around the presence of processes Q and R, which denote the context into which the actual instance of the rule has to be mapped into. In general terms, the need of such a rich contextual information makes more difficult to obtain a satisfying observational semantics. After the initial attempt by Cardelli and Gordon [16], and an early proposal by Ferrari, Montanari and Tuosto [12] exploiting a graphical encoding of processes, we are aware of the work by Merro and Zappa-Nardelli [18] and, quite recently, by Rathke and Sobociński [25].

A series of papers recently addressed the need of synthesising a LTS out of the reduction semantics of e.g. a calculus. The most successful technique so far has been proposed by Leifer and Milner with the so-called *relative pushout* (RPO) [17], which captures in an abstract setting the intuitive notion of minimal context into which a process has to be inserted, in order for allowing a reduction to occur.

However, proving that a calculus satisfies the requirements needed for applying the RPOs technique is often quite a daunting task, due to the intricacies of the structural congruence. A way out of the empasse is represented by looking for graphical encodings of processes, such that process congruence is turned into graph isomorphism. Graphs are amenable to the RPOs trappings, and once the processes of a calculus have been encoded as graphs, a suitable LTS can be distilled. Indeed, the main source of examples concerning RPOs have been *bigraphs* [21], a graphical formalism introduced by Milner for specifying concurrent and distributed systems.

It is noteworthy that, should the reduction relation over graphs be defined using the double pushout (DPO) approach [1], these graphs are amenable to the *borrowed context* (BC) technique, developed by Ehrig and König, which offers an algorithmic solution for calculating the minimal contexts enabling a graph transformation rule

So, the approach pursued in this and other papers [4,14,15] is straightforward: for a given calculus, a graphical encoding (over standard graphs) is found such that process congruence is preserved, and the reduction semantics is captured by a set of graph transformation rules, specified using the DPO approach. A LTS for the calculus is thus immediately distilled. Indeed, this is the way which allowed to derive the unique successful application so far of the RPO technique to the set of recursive processes of a calculus, still recovering the standard bisimulation congruence, even if for admittedly one of simplest calculus available, namely, Milner's CCS [19].

This paper exploits a graphical encoding for MAs [13] to distill a LTS on (processes encoded as) graphs, and a set of inference rules on the processes of the MAs calculus. We compare these rules with alternative proposals, discovering many similarities (thus confirming the hints provided by the ingenuity of the researchers), yet with a few substantial differences, as articulated in the concluding section. Since we are interested in LTS defined over processes, we provide a comparison with the only two works presenting a LTS on MAs processes, namely, those proposed in [18,25].

This paper is organized as follows. Section 2 briefly recalls the MAs calculus. In Section 3 we introduce (typed hyper-)graphs and their extension with interfaces, while Section 4 presents DPO rewriting on graphs with interfaces as well as the BC technique for distilling a LTS. Then, in Section 5 we recall a graphical encoding for MAs processes that has been introduced in [13]. A graph transformation system for MAs that simulates process reduction is defined in Section 6. Section 7 presents a LTS for graphs representing MAs processes, obtained by means of the BC synthesis

mechanism. Section 8 introduces a LTS defined over processes of the MAs calculus and obtained from the LTS over graphs. Finally, Section 9 concludes the paper.

An extended version of this paper is available as [5].

2 Mobile Ambients

This section shortly recalls the finite, communication-free fragment of mobile ambients [8], its structural equivalence and the associated reduction semantics.

Table 1 shows the syntax of the calculus. We assume a set $\mathcal{N}$ of *names* ranged over by $m, n, u, \ldots$. Besides the standard constructors, we included a set of *process variables* $\mathcal{X} = \{X, Y, \ldots\}$, and a set of *name variables* $\mathcal{V} = \{x, y, \ldots\}$. Intuitively, an extended process such as $x[P]|X$ represents an underspecified process, where either the process X or the name of the ambient $x[-]$ can be further instantiated. These are needed for the presentation of the LTS in Section 8.

$$P ::= \mathtt{0}, n[P], M.P, (\nu n)P, P_1|P_2, X, x[P] \qquad M ::= in\ n, out\ n, open\ n$$

Table 1
(Extended) Syntax of mobile ambients.

We use the standard definitions for the set of free names of a process P, denoted by $fn(P)$, and for α-convertibility, with respect to the restriction operators (νn). We let $P, Q, R, \ldots$ range over the set $\mathcal{P}$ of *pure* processes, i.e., such that neither process nor name variable is contained. While $P_\epsilon, Q_\epsilon, R_\epsilon, \ldots$ range over the set $\mathcal{P}_\epsilon$ of *well-formed* processes, i.e., such that no process or ambient variable occurs twice.

We also consider a family of *substitutions*, which may replace a process/name variable with a pure process/name, respectively. Substitutions avoid name capture: for a pure process P, the expression $(\nu n)(\nu m)(X|x[\mathtt{0}])\{^{m}/_{x}, ^{n[P]}/_{X}\}$ corresponds to the pure process $(\nu p)(\nu q)(n[P]|m[\mathtt{0}])$, for names $p, q \notin \{m\} \cup fnn[P]$.

The semantics of the calculus is given by means of a reduction relation and a *structural congruence*, denoted by $\equiv$, which is the least equivalence on pure processes that satisfies the equations and the rules shown in Table 2. The congruence relates processes which intuitively specify the same system, up-to a syntactical rearrangement of its components, and it is then used to define the operational semantics.

The *reduction relation*, denoted by $\rightarrow$, describes the evolution of processes over time: $P \rightarrow Q$ means that P reduces to Q, that is, P can execute a computational step and it is transformed into Q. Table 3 shows the reduction rules.

The reduction relation is closed with respect to structural congruence. Note that our chosen congruence slightly differs from the standard one, since we drop the axiom $(\nu n)\mathtt{0} \equiv \mathtt{0}$, and we add $(\nu n)M.P \equiv M.(\nu n)P$, allowing a restriction to enter a capability. The reduction semantics does not substantially change. Indeed, the equality induced by the latter axiom holds in the observational equivalence proposed by Merro and Zappa Nardelli [18]. In particular, two processes that are structurally congruent according to the axiom *Cong-Res-Act* are *reduction barbed* congruent.

$P\|Q \equiv Q\|P$	$(P\|Q)\|R \equiv P\|(Q\|R)$
$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	$(\nu n)(P\|Q) \equiv P\|(\nu n)Q$ if $n \notin fn(P)$
$P \equiv Q \Rightarrow n[P] \equiv n[Q]$	$(\nu n)m[P] \equiv m[(\nu n)P]$ if $n \neq m$
$P \equiv Q \Rightarrow M.P \equiv M.Q$	$P\|\mathtt{0} \equiv P$
$P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$	$(\nu n)M.P \equiv M.(\nu n)P$ if $n \notin fn(M)$
$P \equiv Q \Rightarrow P\|R \equiv Q\|R$	$(\nu n)P \equiv (\nu m)(P\{^m/_n\})$ if $m \notin fn(P)$

Table 2
Structural congruence on pure processes.

$n[in\ m.P\|Q]\|m[R] \to m[n[P\|Q]\|R]$	$P \to Q \Rightarrow (\nu n)P \to (\nu n)Q$
$m[n[out\ m.P\|Q]\|R] \to n[P\|Q]\|m[R]$	$P \to Q \Rightarrow n[P] \to n[Q]$
$open\ n.P\|n[Q] \to P\|Q$	$P \to Q \Rightarrow P\|R \to Q\|R$

Table 3
Reduction relation on pure processes.

3 Graphs and Their Extension with Interfaces

We recall a few definitions concerning (typed hyper-)graphs, and their extension with *interfaces*, referring to [9] for a more detailed introduction.

Definition 3.1 (graphs) *A* (hyper-)graph *is a four-tuple* $\langle V, E, s, t\rangle$ *where* V, E *are the sets of nodes and edges and* $s, t : E \to V^*$ *are the source and target functions. A* graph morphism *is a pair of functions* $\langle f_V, f_E\rangle$ *preserving source and target.*

The corresponding category is denoted by **Graph**. However, we often consider *typed graphs* [10], i.e., graphs labeled over a structure that is itself a graph.

Definition 3.2 (typed graphs) *Let* T *be a graph. A* typed graph G *over* T *is a graph* $|G|$*, together with a graph morphism* $t_G : |G| \to T$*. A* T-typed graph morphism *is a graph morphism* $f : |G_1| \to |G_2|$ *preserving the typing.*

The category of graphs typed over T is denoted T-**Graph**.

Definition 3.3 (graphs with interfaces) *Let* J, K *be typed graphs. A* graph with input interface J and output interface K *is a triple* $\mathbb{G} = \langle j, G, k\rangle$*, for* G *a typed graph and* $j : J \to G$*,* $k : K \to G$ *the* input *and* output *morphisms.*

Let $\mathbb{G}$ *and* $\mathbb{H}$ *be graphs with the same interfaces. An* interface graph morphism $f : \mathbb{G} \Rightarrow \mathbb{H}$ *is a typed graph morphism* $f : G \to H$ *between the underlying graphs that preserves the input and output morphisms.*

We let $J \xrightarrow{j} G \xleftarrow{k} K$ denote a graph with interfaces J and K.[2] If the interfaces J, K are *discrete*, i.e., they contain only nodes, we represent them by sets; if K is the empty set, we often denote a graph with interfaces as a graph morphism $J \to G$.

[2] With an abuse of terminology, we sometimes refer to the image of the input and output morphisms as inputs and outputs, respectively. Thus, in the following we often refer implicitly to a graph with interfaces as the representative of its isomorphism class, still using the same symbols to denote it and its components.

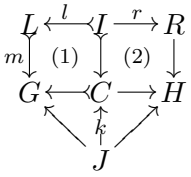
In order to define a process encoding, some (binary) operators on graphs with discrete interfaces should be defined. Since we rely on the encoding presented in [13], we refer the reader there for details, and to [5, Appendix A] for a quick survey.

4 On Graphs with Interfaces and Borrowed Contexts

This section introduces the *double-pushout* (DPO) approach to the rewriting of graphs with interfaces and its extension with *borrowed contexts* (BCs).

Definition 4.1 (graph production) *A T-typed* graph production *is a span $L \xleftarrow{l} I \xrightarrow{r} R$ with l mono in T-***Graph***. A T-typed* graph transformation system *(GTS) $\mathcal{G}$ is a pair $\langle P, \pi \rangle$ where P is a set of production names and π assigns each production name to a T-typed production.*

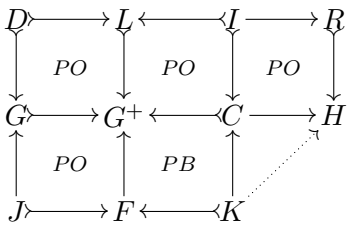
Definition 4.2 (derivation of graphs with interfaces) *Let $J \to G$ and $J \to H$ be two graphs with interfaces. Given a production $p : L \leftarrowtail I \longrightarrow R$, a* match *of p in G is a mono $m : L \rightarrowtail G$. A* direct derivation *from $J \to G$ to $J \to H$ via p and m is a diagram as depicted in the right, where (1) and (2) are pushouts and the bottom triangles commute. In this case we write $J \to G \Longrightarrow J \to H$.*



The morphism $k : J \to C$ which makes the left triangle commute is unique, whenever it exists. If such a morphism does not exist, then the rewriting step is not feasible. Moreover, note that the standard DPO derivations can be seen as a special instance of these, obtained considering as interface J the empty graph.

In these derivations, the left-hand side L of a production must occur completely in G. In a *borrowed context* (BC) derivation the graph L might occur partially in G, since the latter may interact with the environment through J in order to exactly match L. Those BCs are the "smallest" extra contexts needed to obtain the image of L in G. The mechanism was introduced in [11] in order to derive a LTS from direct derivations, using BCs as labels. The following definition is lifted from [24], extended by including morphisms that are not necessarily mono.

Definition 4.3 (rewriting with borrowed contexts) *Given a production $p : L \xleftarrow{l} I \xrightarrow{r} R$, a graph with interfaces $J \to G$ and a mono $d : D \rightarrowtail L$, we say that $J \to G$* reduces to *$K \to H$ with transition label $J \rightarrowtail F \leftarrowtail K$* via *$p$ and d if there are graphs G^+, C and additional morphisms such that the diagram below commutes and the squares are either pushouts (PO) or pullbacks (PB). In this case we write $J \to G \xrightarrow{J \rightarrowtail F \leftarrowtail K} K \to H$, also called* rewriting step with borrowed context.



Consider the diagram above. The upper left-hand square merges the left-hand side L and the graph G to be rewritten according to a partial match $G \leftarrowtail D \rightarrowtail L$.

The resulting graph G^+ contains a total match of L and can be rewritten as in the standard DPO approach, producing the two remaining squares in the upper row. The pushout in the lower row gives the borrowed context F which is missing in order to obtain a total match of L, along with a morphism $J \rightarrowtail F$ indicating how F should be pasted to G. Finally, the interface for the resulting graph H is obtained by "intersecting" the borrowed context F and the graph C via a pullback.

Note that two pushout complements that are needed in Definition 4.3, namely C and F, may not exist. In this case, the rewriting step is not feasible.

5 Graphical Encoding for Processes

This section shortly recalls a graphical encoding for MAs processes. After the description a type graph (T_M, depicted in Figure 1), the encoding is defined by means of suitable operators on typed graphs with interfaces. This corresponds to a variant of the usual construction of the tree for a term of an algebra: names are interpreted as variables, so they are mapped to graph leaves and can be shared.

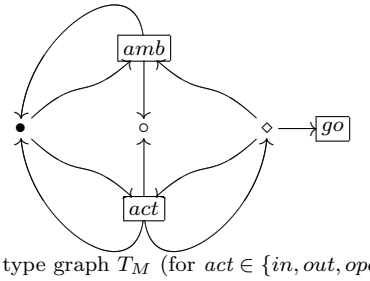


Fig. 1. The type graph T_M (for $act \in \{in, out, open\}$).

Intuitively, a node of type $\circ$ represents an ambient name, while a graph that has as roots a pair of nodes $\langle \diamond, \bullet \rangle$ represents a process. More precisely, the node of type $\diamond$ represents the activating point for reductions of the process represented by the graph. We need two different types of node to model processes by graphs, because each graph has to model both syntactical dependences between the operators of the process and their activation dependences.

Each edge of the type graph, except the *go* edge, simulates an operator of MAs. Note that the *act* edge represents three edges, namely *in*, *out* and *open*. These edges simulate the capabilities of the calculus, while the *amb* edge simulates the ambient operator, and there are no edges to simulate the restriction operator and the parallel composition. Finally, the *go* edge is a syntactical device for detecting the "entry" point for the computation. We need it later to simulate MAs reduction semantics. It allows to avoid the occurrence of a reduction underneath a *act* operator.

We remark that choosing a graph typed over T_M means to consider graphs where each node (edge) is labeled by a node (edge) of that type graph, and the incoming and outcoming tentacles are preserved. We refer the reader to [13] for the formal presentation of the encoding, or to [5, Appendix B] for a short recollection.

For our purposes it then suffices to say that the encoding $[\![P]\!]^{go}_{\Gamma}$ of a pure process P, where Γ is a set of names such that $fn(P) \subseteq \Gamma$, is a graph with interfaces $(\{a, p\} \cup \Gamma, \emptyset)$, for $a, p \notin \mathcal{N}$. Our encoding is sound and complete with respect to the structural congruence $\equiv$, as stated by the proposition below.

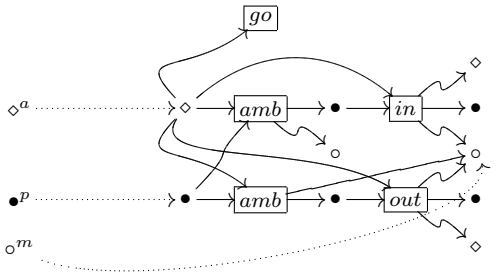


Fig. 2. Graph encoding for the process $(\nu n)(n[in\ m.0]|m[out\ m.0])$.

Proposition 5.1 *Let P, Q be pure processes and let Γ be a set of names, such that $fn(P) \cup fn(Q) \subseteq \Gamma$. Then, $P \equiv Q$ if and only if $[\![P]\!]^{go}_{\Gamma} = [\![Q]\!]^{go}_{\Gamma}$.*

Example 5.2 Consider the pure process $P = (\nu n)(n[in\ m.0]|m[out\ m.0])$. It is a very simple process, which represents a restricted ambient n that can enter an ambient m. Figures 2 depicts the graph encoding for the process above. The leftmost edges, both labeled *amb*, have the same roots, into which the nodes of the interface a and p are mapped. Those two edges represent the topmost operators of the two parallel components of the process. The edge *in* represents the operator $in\ m$ that is inside the restricted ambient n, while the edge *out* represents the operator $out\ m$ that is inside the ambient m. These two last edges are linked to the same root node $\diamond$ of their parent ambients. Intuitively, this means that they can be involved in a reduction step, too, since the only edge labeled *go* is linked to that same node. Note that the ambient name m is in the interface since it is free in P, instead the name n, which is bound, does not belong to the interface.

6 Graph Transformation for Mobile Ambients

This section presents a graph transformation system (GTS) that models the reduction semantics of the MAs calculus.

Figure 3 presents the rules of the GTS $\mathcal{R}_{amb}$, which simulates the reduction semantics $\rightarrow$ introduced in Section 2. The GTS $\mathcal{R}_{amb}$ contains just three rules, namely p_{in}, p_{out} and p_{open}. They simulate the three axioms of the reductions relation. The action of the three rules is described by the node identifiers. These identifiers are of course arbitrary: they correspond to the actual elements of the set of nodes and are just used to characterize the span of functions.

It seems noteworthy that three rules [3] suffice for recasting the reduction semantics of mobile ambients. That is possible for two reasons. First, the closure of reduction with respect to contexts is obtained by the fact that graph morphisms allow the embedding of a graph within a larger one. Second, no distinct instance of the rules is needed, since graph isomorphism takes care of the closure with respect to structural congruence, and interfaces of the renaming of free names.

Our encoding is sound and complete with respect to the reduction relation $\rightarrow$, as stated by the theorem below.

[3] Actually, five: since we consider mono matches, we need to assume an instance for the rules p_{in} and p_{out}, where the nodes labeled n and m may actually be coalesced

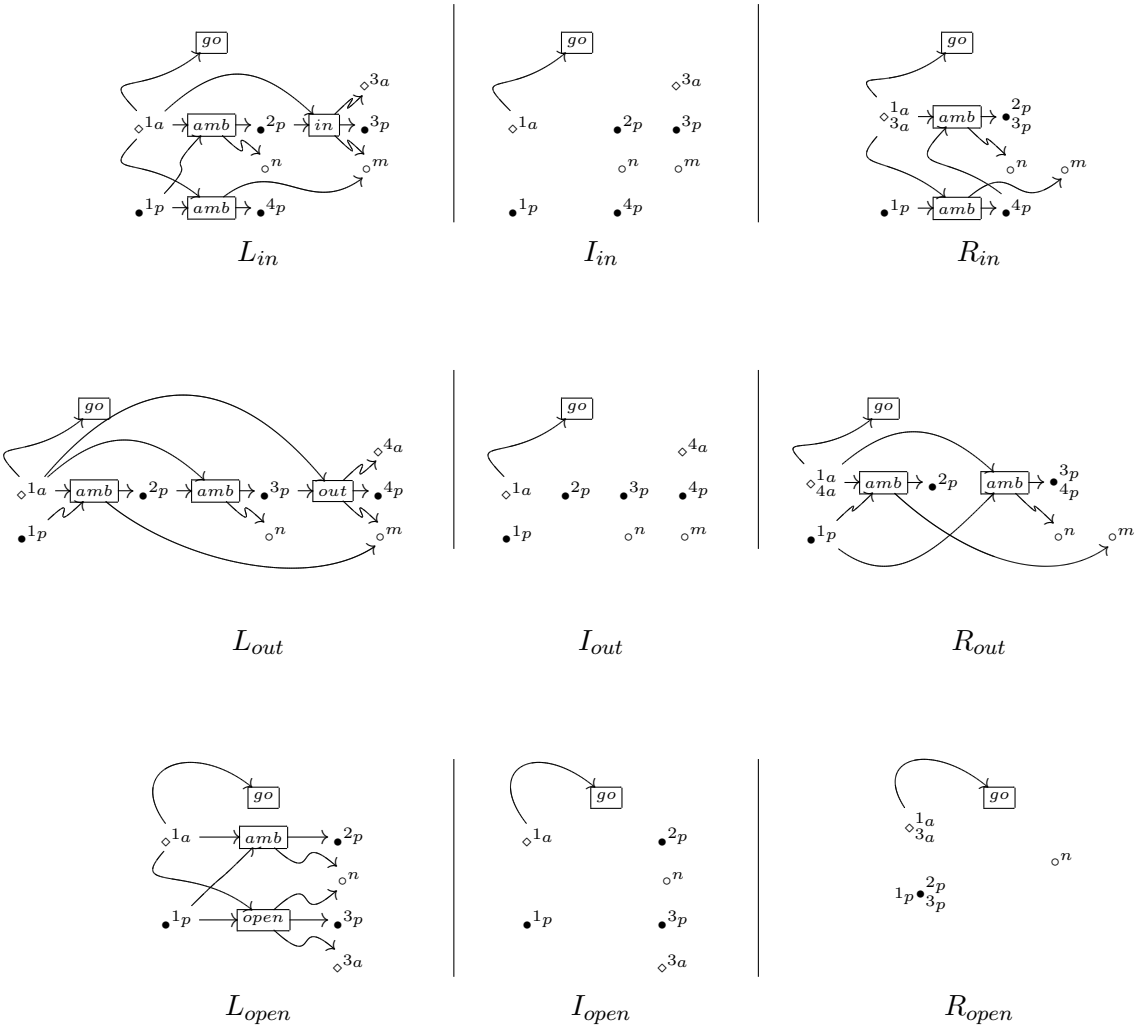


Fig. 3. The rewriting rules p_{in}, p_{out} and p_{open}.

Theorem 6.1 (reductions vs. rewrites) *Let P be a pure process, and let Γ be a set of names, such that $fn(P) \subseteq \Gamma$. If $P \to Q$, then $\mathcal{R}_{amb}$ entails a direct derivation $[\![P]\!]^{go}_{\Gamma} \Longrightarrow [\![Q]\!]^{go}_{\Gamma}$. Vive versa, if $\mathcal{R}_{amb}$ entails a direct derivation $[\![P]\!]^{go}_{\Gamma} \Longrightarrow \mathbb{G}$, then there exists a pure process Q, such that $P \to Q$ and $\mathbb{G} = [\![Q]\!]^{go}_{\Gamma}$.*

The correspondence holds since a rule is applied only if there is a match that covers a subgraph with the *go* operator on the top. This allows the occurrence of reductions inside activated ambients, but not inside capabilities.

7 The Synthesized Transition System

In this section we apply the BC synthesis mechanism to $\mathcal{R}_{amb}$ in order to obtain a LTS for graphs representing MAs processes. We first show some examples of rewriting steps with BCs, then we use some pruning techniques (proposed in [4]) in order to obtain a simpler presentation of the derived LTS. This presentation is then used in the next section in order to define a LTS directly over MAs.

7.1 Examples of borrowed transitions

This section shows the application of the BC synthesis mechanism to the graphical encoding of a process. Let us consider the graph $J \rightarrowtail G = [\![P]\!]^{go}_{\{m\}}$, where $P = (\nu n)(n[in\ m.0]|m[out\ m.0])$. In the following we discuss the possible transitions with source $J \rightarrowtail G$ that are induced by the rule $p_{in} : L_{in} \leftarrowtail I_{in} \rightarrow R_{in}$ of $\mathcal{R}_{amb}$. Since for each pair of monos $G \leftarrowtail D \rightarrowtail L_{in}$ a labeled transition might exist, we proceed by showing the transitions generated by such pairs.

First of all, take as D the left-hand side L_{in} and note that there is only one map into the graph G. (The transition generated by this choice is depicted in [5, Figure 8].) The graph G^+ is the same as G. Now C and H are constructed as in a standard DPO rewriting step. When taking D as the whole left-hand side, $J \rightarrowtail G$ needs no context for the reaction and thus the label of this transition is the identity context, i.e., two isomorphisms into the discrete graphs with three nodes $\{p, a, m\}$. Intuitively, this corresponds to an internal transition over processes, labeled with τ.

Now we take as D the subgraph of L_{in} representing an ambient with a capability in inside it. Note that also in this case there is only one possible map into the graph G. (The resulting transition is shown in [5, Figure 9]. The graph G^+ is the graph G in parallel with the graph representing an ambient m, thus intuitively it represents the process $(\nu n)(n[in\ m.0]|m[out\ m.0]|m[X])$ for some process variable X. The graph $J \rightarrowtail G$, in order to reach the graph G^+, has to borrow from the environment the context $J \rightarrowtail F \leftarrowtail K$ that represents the syntactic context $-|m[X]$. Note that in the resulting interface K there is a process node $\bullet^{4_p}$ pointing to the process node of F occurring inside the ambient m. This process node in K represents the process variable X (as further detailed in [5, Appendix E]). The graphs C and H are then constructed as in the standard DPO approach. Intuitively, $K \rightarrow H$ represents the process $m[out\ m.0]|m[n[0]|X]$, where X is the same process variable occurring in the label $J \rightarrowtail F \leftarrowtail K$. This can be understood by observing that the process node $\bullet^{4_p}$ of K points both to a node of H and to a node of F. Summarizing, this transition moves the ambient n into an ambient m that is provided by the environment.

Another possible D is the subgraph of L_{in} consisting of the ambient depicted in the lower part of L_{in}. In this case, there are two possible maps into the graph G: the map into the subgraph of G representing the ambient m, and the map into the subgraph of G representing the restricted ambient n.

In the first case, we obtain the transition (shown in [5, Figure 10]) where the graph G^+ is the graph G in parallel with the graph representing a fresh ambient name w having inside a capability $in\ m$. Intuitively, it represents the extended process $(\nu n)(n[in\ m.0]|m[out\ m.0]|w[in\ m.X_2|X_1])$ for some process variables X_1, X_2. In order to reach G^+, the graph $J \rightarrowtail G$ has to borrow from the environment the context $J \rightarrowtail F \leftarrowtail K$ representing the syntactic context $-|w[in\ m.X_2|X_1]$. As in the above case X_1 and X_2 are process variables, since in the interface K there are two suitable process nodes $\bullet^{2_p}$ and $\bullet^{3_p}$. The graphs C and H are obtained by a standard DPO derivation. The graph $K \rightarrow H$ represents the extended process $(\nu n)(n[in\ m.0]|m[out\ m.0|w[X_2|X_1]])$. Summarizing, this transition represents an ambient w from the environment entering inside the ambient m of the process P.

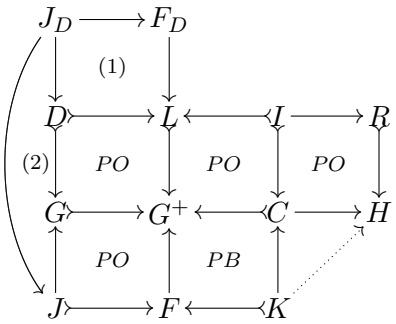


Fig. 4. The BC construction together with commuting squares (1) (the initial pushout of $D \rightarrowtail L$) and (2).

In the second case no transition is possible. Indeed the graph G^+ is the whole graph G in parallel with a fresh ambient w having inside a capability *in* n, but the pushout complement of $J \rightarrowtail G \rightarrowtail G^+$ does not exist, because n is restricted and thus it does not belong to the interface J. Intuitively, this means that no ambient from the environment can enter inside a restricted sibling ambient n.

In order to perform a complete analysis, we should consider all the pairs of monos $G \leftarrowtail D \rightarrowtail L_{in}$: we can avoid to check the others pairs not considered above by exploiting the pruning techniques presented in the next subsection.

7.2 Reducing the borrowing

In order to know all the possible transitions originating from a graph with interfaces $J \rightarrowtail G$, all the subgraphs D's of L_{in}, L_{out} and L_{open} should be analyzed. To shorten this long and tedious procedure, we use the two pruning techniques presented in [4].

The first one is based on the observation that those items of a left-hand side L that are not in D have to be glued to G through J. Let us consider a node n of D corresponding to a node n' in L, such that n' is the source or the target of some edge e that does not occur in D. Since the edge e is in L but not in D, it must be added to G through J, and thus n, called *boundary node*, must be also in J.

The notion of boundary nodes is formally captured by the categorical notion of *initial pushout* (as defined e.g. in [5, Appendix C]). Since our category has initial pushouts, the previous discussion is formalized by the lemma below.

Lemma 7.1 ([4]) *A graph with interfaces $J \rightarrow G$ can perform a BC rewriting step in $\mathcal{R}_{amb}$ if and only if there exist*

- *a mono $D \rightarrowtail L$ (where L is the left hand side of some production in $\mathcal{R}_{amb}$),*
- *a mono $D \rightarrowtail G$,*
- *a morphism $J_D \rightarrow J$ (where J_D is the initial pushout of $D \rightarrowtail L$) such that square (2) in Figure 4 commutes.*

This corollary allows to heavily prune the space of possible D's. As for graphs corresponding to the encoding of processes, we can exclude all those D's having a continuation process node (any node depicted by $\bullet$ that is not the root) as boundary node, observing that the only process node in the interface J is the root node.

A further pruning —partially based on proof techniques presented in [11]— is performed by excluding all those D's which generate a BC transition that is not relevant for the bisimilarity. In general terms, we may exclude all the D's that contain only nodes, since those D's can be embedded in every graph (with the same interface) generating the same transitions. Moreover, concerning our case study, those transitions generated by a D having the root node without the edge labeled *go* are also not relevant. In fact, a graph can perform a BC transition using such a D if and only if it can perform a transition using the same D with a *go* edge outgoing from the root. Note indeed that the resulting states of these two transitions only differ for the number of *go* edges attached to the root: the state resulting after the first transition has two *go*'s, the state resulting after the second transition only one. These states are bisimilar, since the number of *go*'s does not change the behavior.

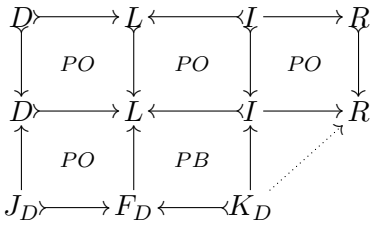
The pruning techniques above allow us to consider a restricted set of partial matches D, namely, the ones shown in Figure 5 and in [5, Figures E.1, E.2, E.3].

7.3 Minimal transitions

In Section 7.2 we restricted the space of possible D's. However, reasoning on the synthesized LTS is still hard (this is usually the case when working with derived LTSs, as pointed out in [2] and [3], where the authors state that an SOS presentation of the synthesized LTS would be desirable). In order to simplify this reasoning, we introduce a set of *minimal transitions* that allow us to derive all and only the transitions of the (pruned) synthesized LTS.

Inspired by Lemma 7.1, providing necessary and sufficient conditions for performing a transition, we consider the graphs $J_D \to D$ for all those D's that have not been pruned in Section 7.2 and J_D containing only the boundary nodes of D.

The minimal transitions have the following shape



where the leftmost square in the lower row is an initial pushout.

Figure 5 concisely represents such a transition, showing the starting graph D, the label $J_D \rightarrowtail F_D \leftarrowtail K_D$, and the resulting graph R. All the transitions that are originated from a graph $J \rightarrowtail G$ (representing a process) can be characterized by exploiting these minimal transitions. By Lemma 7.1, we can state that $J \rightarrowtail G$ can perform a BC rewriting step in $\mathcal{R}_{amb}$ if and only if there exist a mono $D \rightarrowtail G$, for some D of the minimal transitions, and a morphism $J_D \to J$ such that square (2) in Figure 4 commutes.

The label of the rewriting step can be obtained from the label of the minimal transition. First of all note that the interface J contains all the nodes of J_D (as suggested by the morphism $J_D \to J$) and all the name nodes $\circ$ representing the free names of the modeled process (as expected by our encoding). Then the graph

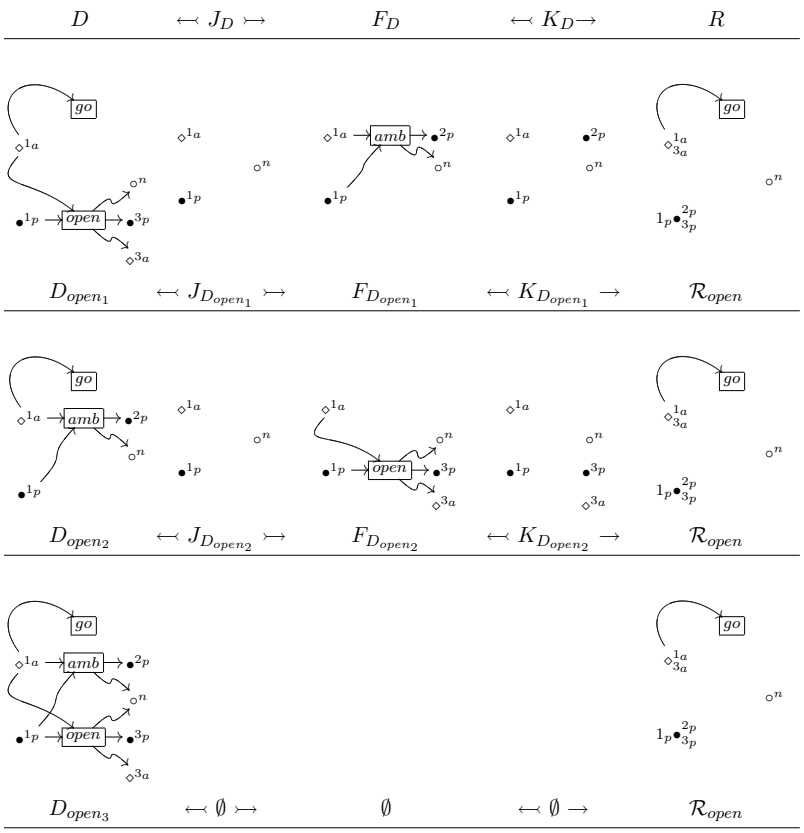


Fig. 5. The minimal transitions generated by the rule p_{open}.

F only contains the whole graph F_D and all the nodes of J (indeed, as shown in [4, Proposition 2.5], F can be obtained as the pushout of $J_D \to F_D$ and $J_D \to J$). Moreover, it is easy to prove that K is a discrete graph containing all and only the nodes of F, or more concretely, K consists of the nodes of J and K_D.

Finally, the resulting graph H is obtained by replacing in the graph G the subgraph D with R (as shown in Proposition 2.5 of [4], it can be computed in a DPO step of $D \leftarrowtail D \cap I \to R$, where $D \cap I$ is the pullbacks of $D \rightarrowtail L$ and $I \rightarrowtail L$).

8 A New LTS for Mobile Ambients

This section presents a LTS directly defined over MAs processes. The inference rules describing this LTS are obtained from the transitions of the LTS on graphs presented in Section 7.3. The labels of the transitions are unary contexts, i.e., terms of the extended syntax with a hole $-$. The formal definition of our LTS is presented in Figures 6 and 7.

8.1 The labeled rules on processes...

The rules in Figure 6 represent the τ-actions modeling internal computations. Note that the labels of the transitions are contexts composed of just a hole $-$, while the resulting states are processes over MAs standard syntax. The rule InTau enables an ambient n to enter a sibling ambient m. The rule OutTau enables an ambient n

to get out of its parent ambient m. Finally, the rule OpenTau models the opening of an ambient n. These three rules exactly derive the same transition relation of the reduction relation over MAs, thus they could be replaced with the rules in Table 3.

The rules in Figure 7 model the interactions of a process with its environment. Note that both labels and resulting states contain process and name variables. We define a LTS for processes over the standard syntax of mobile ambients by instantiating all the variables of the labels and of the resulting states. Formally, we say that $P \xrightarrow{l} Q$ (for l and Q pure processes) if and only if $P \xrightarrow{l_\epsilon} Q_\epsilon$ and there exists a substitution σ such that $Q_\epsilon\sigma \equiv Q$ and $l_\epsilon\sigma \equiv l$.

The rule Open models the opening of an ambient provided by the environment. In particular, it enables a process P with a capability $open\ n.P_1$ at top level, for $n \in fn(P)$, to interact with a context providing an ambient n that contains inside it some process X_1. The resulting state is the process over the extended syntax $(\nu A)(P_1|X_1|P_2)$, where X_1 represents a process provided by the environment. Note that the instantiation of the process variable X_1 with a process containing a free name that belongs to the bound names in A is possible only α-converting the resulting process $(\nu A)(P_1|X_1|P_2)$ into a process that does not contain that name among its bound names at top level.

The rule CoOpen instead models an environment that opens an ambient of the process. The rule InAmb enables an ambient of the process to migrate into a sibling ambient provided by the environment, while in the rule In both the ambients are provided by the environment. In the rule CoIn an ambient provided by the environment enters an ambient of the process. The rule OutAmb models an ambient of the process exiting from an ambient provided by the environment, while in the rule Out both ambients are provided by the environment.

Our LTS does not conform to the so-called SOS style: indeed, the premises of the inference rules are just constraints over the structure of the process. This depends on fact that the rules of our LTS are obtained from the borrowed minimal transitions. Each rule corresponds to one minimal transition presented in Section 7.3 and it is obtained as described below.

$$\text{(InTau)}\ \frac{P\equiv(\nu A)\ \mathcal{C}[n[in\ m.P_1|P_2]|m[P_3]]}{P\xrightarrow{-}(\nu A)\ \mathcal{C}[m[n[P_1|P_2]|P_3]]} \qquad \text{(OutTau)}\ \frac{P\equiv(\nu A)\ \mathcal{C}[m[n[out\ m.P_1|P_2]|P_3]]}{P\xrightarrow{-}(\nu A)\ \mathcal{C}[m[P_3]|n[P_1|P_2]]}$$

$$\text{(OpenTau)}\ \frac{P\equiv(\nu A)\ \mathcal{C}[n[P_1]|open\ n.P_2]}{P\xrightarrow{-}(\nu A)\ \mathcal{C}[P_1|P_2]}$$

Fig. 6. The internal transitions (for $\mathcal{C}[-]$ context containing only ambients and parallel operators).

8.2 ...from the borrowed rules on graphs

Observe that a graph $J \rightarrowtail G$ representing a process P can perform a BC rewriting step in $\mathcal{R}_{amb}$ if and only if there exist a mono $D \rightarrowtail G$, for some D of a minimal transition, and a morphism $J_D \to J$, such that square (2) in Figure 4 commutes. Moreover, the label and the resulting graph of the borrowed transition for G are obtained from the label and the resulting state of the minimal transition of D,

$$\text{(In)}\ \frac{P \equiv (\nu A)(in\ m.P_1|P_2) \quad m \notin A}{P \xrightarrow{x[-|X_1]|m[X_2]} (\nu A)m[x[P_1|P_2|X_1]|X_2]} \qquad \text{(OutAmb)}\ \frac{P \equiv (\nu A)(n[out\ m.P_1|P_2]|P_3) \quad m \notin A}{P \xrightarrow{m[-|X_1]} (\nu A)(m[P_3|X_1]|n[P_1|P_2])}$$

$$\text{(InAmb)}\ \frac{P \equiv (\nu A)(n[in\ m.P_1|P_2]|P_3) \quad m \notin A}{P \xrightarrow{-|m[X_1]} (\nu A)(m[n[P_1|P_2]|X_1]|P_3)} \qquad \text{(Open)}\ \frac{P \equiv (\nu A)(open\ n.P_1|P_2) \quad n \notin A}{P \xrightarrow{-|n[X_1]} (\nu A)(P_1|P_2|X_1)}$$

$$\text{(CoIn)}\ \frac{P \equiv (\nu A)(m[P_1]|P_2) \quad m \notin A}{P \xrightarrow{-|x[in\ m.X_1|X_2]} (\nu A)(m[x[X_1|X_2]|P_1]|P_2)} \qquad \text{(CoOpen)}\ \frac{P \equiv (\nu A)(n[P_1]|P_2) \quad n \notin A}{P \xrightarrow{-|open\ n.X_1} (\nu A)(P_1|X_1|P_2)}$$

$$\text{(Out)}\ \frac{P \equiv (\nu A)(out\ m.P_1|P_2) \quad m \notin A}{P \xrightarrow{m[x[-|X_1]|X_2]} (\nu A)(m[X_2]|x[P_1|P_2|X_1])}$$

Fig. 7. The environmental transitions.

respectively. Therefore, for each minimal transition we obtain an inference rule: the conditions in the premise correspond to the necessary and sufficient conditions for performing a transition from a graph G, while the label and the resulting process are obtained from the label and the resulting state of the borrowed transition, respectively. Since the labels of the LTS over graphs obtained by the BC mechanism represent minimal graph contexts enabling a graph production, then also the labels of our LTS over processes represent minimal process contexts enabling a reduction.

As the main example, in this section we closely look at the correspondence between the rule Open and the first minimal transition in Figure 5.

Consider a graph $J \rightarrowtail G$ representing the encoding for a process P. If there exist a mono $D_{open_1} \rightarrowtail G$ and a morphism $J_{D_{open_1}} \to J$, such that the square (2) in Figure 4 commutes, the graph $J \rightarrowtail G$ can perform a BC rewriting step in $\mathcal{R}_{amb}$ with label $J \rightarrowtail F \leftarrowtail K$, where J, F and K respectively consist of $J_{D_{open_1}}$, $F_{D_{open_1}}$ and $K_{D_{open_1}}$ together with the free names of P. Now, note that D_{open_1} can be embedded in G and a morphism $J_{D_{open_1}} \to J$ (such that the square (2) in Figure 4 commutes) may exist if and only if $P \equiv (\nu A)(open\ n.P_1|P_2)$. Moreover, since the interface J contains all the nodes of $J_{D_{open_1}}$, we conclude that n must belong to J, that is, n must be a free name of P. This represents the premise of the rule Open.

Starting from the label $J \rightarrowtail F \leftarrowtail K$ of the BC transition we now obtain the label of the process transition. By observing the shape of F, which contains all the items of $F_{D_{open_1}}$, we can say that the process context is composed of the ambient n. Moreover, the context F is glued to G through J, which contains the free names of P and the nodes of $J_{D_{open_1}}$, i.e., the name n and the nodes representing the roots of the graph G (which models P). Since these two nodes represent the roots of the graph F (which models ambient n), we conclude that the label of the process transition is a context with the ambient n in parallel with a hole representing process P.

The graph K represents the interface of both graphs F and H. It contains all the nodes of $K_{D_{open_1}}$, i.e., the roots of F and the roots of the process inside the ambient n. The nodes of the interface K represent the "handles" of F and H for interacting with an environment. Therefore, the process node of K that is not the root of F can be thought of as a process variable inside the ambient n in the label of the transition. Therefore, we conclude that the label of the transition with source the

process P can be represented as the minimal context $-|n[X_1]$, where $-$ is a hole and X_1 is a process variable. The resulting process $(\nu A)(P_1|X_1|P_2)$ exactly corresponds to the state H from the BC transition. Indeed, in the interface K of the graph $K \to H$ also the node modeling the process variable X_1 occurs, which represents a process provided by the environment. In order to have a deeper intuition about the correspondence between process variables and graphs, the reader is referred to [5, Appendix E]. The derivation of the rule Out is shown in [5, Appendix D].

9 Conclusions, related and future work

In this paper we exploit the graphical encoding for MAs, proposed in [13], to distill a LTS on (processes encoded as) graphs. We then use this LTS in order to infer a LTS directly defined on the processes of the MAs calculus. For the sake of simplicity, we considered a graphical encoding for MAs without communication primitives, as well as without recursive expressions. A graphical encoding for the whole calculus could be obtained by tackling both communication primitives and recursive processes along the lines of the solution in [4]. Once the graphical encoding for the whole calculus has been defined, the technique presented in this paper could be applied in order to obtain a LTS for the whole MAs calculus.

In spite of the great interest received by MAs, there are relatively few works concerning a labeled characterization of the calculus. After early attempts by Cardelli and Gordon [16] and (*via* a gfraphical encoding) Ferrari, Montanari and Tuosto [12], the only papers addressing this issue that we are aware of are [18] by Merro and Zappa-Nardelli, [25] by Rathke and Sobociński. The LTS of the former work is restricted to *systems*, i.e., those processes obtained by the parallel composition of ambients. For this reason, our rules In, Open and Out have not a counterpart in [18]. Instead, the rules InAmb, CoIn and OutAmb exactly correspond to the rules (Enter), (Co-Enter), (Exit) in Table 6 of [18]. Moreover, our rule CoOpen roughly corresponds to their (Open). Indeed the former inserts a process into the context $-|\mathit{open}\ n.X_1$, while the latter into $k[-|\mathit{open}\ n.X_1|X_2]$ (again, this difference is due to the fact that the LTS of [18] is restricted to systems).

It is important to note that, differently from our LTS, the labels of the rules (Enter) and (Exit) contain the name of the migrating ambient n. This requires defining two extra rules (Enter Shh) and (Exit Shh) for the case when n is restricted.

Analogously to our work, Rathke and Sobociński employ a general systematic procedure for deriving LTSs that they have previously introduced in [22]. The detailed comparison is left as future work, but we conjecture that the two LTSs exactly correspond. Indeed, the seven axioms in Figure 6 of [25] are in one to one correspondence with our seven rules in Figure 7. The main difference concerns the derivation procedures that have been employed and the presentations of the resulting LTSs. Theirs is presented in a SOS style (as a result of their procedure), while ours relies on the structural congruence (as a result of the BC mechanism applied to the graphical encoding). Their style carries more information than ours, since it describes the behaviour of each syntactic operator, but our presentation seems more intuitive, since it employs fewer compact rules (10 instead of their 27).

Beside the presentation of a succinct LTS for mobile ambients, our work is a relevant case study for the theory of reactive systems [17]. As already pointed out in the introduction, BC rewriting and bigraphical reactive systems [21] are both instances of this theory. This paper, together with [4], shows that the BCs approach is quite effective in deriving LTS for process calculi.

In particular, this work confirms the advantage of BCs over graphs with interfaces with respect to bigraphs. In bigraphs, all the reduction rules must be ground (i.e., they can not contain process variables). As a result, also the labels and the arriving states of the derived transitions must be ground. Instead, rewriting with BCs allows to employ few non ground rules (as shown in this paper) and thus the resulting transitions have labels and arriving states containing (process and name) variables. This feature was not relevant for calculi such as CCS and π, because the variables in the labels always occur "outside" of the arriving state and thus can be forgotten. As an example, consider the CCS transition $a.b \xrightarrow{-|\bar{a}.Y} b|Y$ derived from the (non ground) reduction rule $a.X|\bar{a}.Y \longrightarrow X|Y$. The behaviour of the process $b|Y$ is trivially equivalent to b: their interaction is basically restricted to processes offering a $\bar{b}$ action, and we can thus avoid to consider Y. Instead, in the case of mobile ambients, the ability of considering non ground states is fundamental, because process variables may occur nested inside ambients in arriving states.

The relevance of this work for the theory of reactive systems is not limited to the above observations. The first author has shown in [3] that in reactive systems the bisimilarity on the derived LTS is usually too strict, while *saturated bisimilarity* (i.e., the bisimilarity over the LTS having all contexts as labels and not just the minimal ones) is often more adequate. This is the case of Logic Programming, open π-calculus [6] and Petri nets [7]. The present work provides a further successful test of the above claim. Indeed, it is easy to see that (the standard notion of) bisimilarity over our LTS is too strict, because it allows to observe the ability of an ambient to migrate, while it should be unobservable, as pointed out in [18]. For this reason, Rathke and Sobociński added two extra-rules to their LTS, while Merro and Zappa Nardelli chose an asymmetric definition of bisimilarity. The latter solution recalls us the *semi-saturated bisimulation* [6]. Instead of requiring that two bisimilar processes must perform transitions with the same label, the definition of semi-saturated bisimulation requires that

$$\text{if } P \xrightarrow{C[-]} P_1 \text{ then } C[Q] \text{ reduces to } Q_1 \text{ and } P_1 \; R \; Q_1.$$

It is worth noting that second and third points of Definition 3.2 in [18] has exactly this shape (the labels $*.enter_n$ and $*.exit_n$ correspond to our contexts $-|n[X_1]$ and $n[-|X_1]$). We leave as future work to exploit this intuition and to check if (semi-)saturated bisimulation on our LTS corresponds to the behavioral equivalence proposed by Merro and Zappa Nardelli.

References

[1] Baldan, P., A. Corradini, H. Ehrig, M. Löwe, U. Montanari and F. Rossi, *Concurrent semantics of algebraic graph transformation*, in: H. Ehrig, H.-J. Kreowski, U. Montanari and G. Rozenberg, editors, *Concurrency, Parallelism, and Distribution*, Handbook of Graph Grammars and Computing by Graph Transformation **3**, World Scientific, 1999 pp. 107–187.

[2] Baldan, P., H. Ehrig and B. König, *Composition and decomposition of DPO transformations with borrowed context*, in: A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro and G. Rozemberg, editors, *Graph Transformation*, Lect. Notes in Comp. Sci. **4178** (2006), pp. 153–167.

[3] Bonchi, F., "Abstract Semantics by Observable Contexts," Ph.D. thesis, Department of Informatics, University of Pisa (2008).

[4] Bonchi, F., F. Gadducci and B. König, *Process bisimulation via a graphical encoding*, in: A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro and G. Rozemberg, editors, *Graph Transformation*, Lect. Notes in Comp. Sci. **4178** (2006), pp. 168–183.

[5] Bonchi, F., F. Gadducci and G. V. Monreale, *Labeled transitions for mobile ambients*, Technical Report TR-08-20, Dipartimento di Informatica, Università di Pisa (2008).

[6] Bonchi, F., B. König and U. Montanari, *Saturated semantics for reactive systems*, in: *Logic in Computer Science* (2006), pp. 69–80.

[7] Bonchi, F. and U. Montanari, *Coalgebraic models for reactive systems*, in: L. Caires and V. Vasconcelos, editors, *Concurrency Theory*, Lect. Notes in Comp. Sci. **4703** (2007), pp. 364–379.

[8] Cardelli, L. and A. Gordon, *Mobile ambients*, Theor. Comp. Sci. **240** (2000), pp. 177–213.

[9] Corradini, A. and F. Gadducci, *An algebraic presentation of term graphs, via gs-monoidal categories*, Applied Categorical Structures **7** (1999), pp. 299–331.

[10] Corradini, A., U. Montanari and F. Rossi, *Graph processes*, Fundamenta Informaticae **26** (1996), pp. 241–265.

[11] Ehrig, H. and B. König, *Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts*, Mathematical Structures in Computer Science **16** (2006), pp. 1133–1163.

[12] Ferrari, G., U. Montanari and E. Tuosto, *A lts semantics of ambients via graph synchronization with mobility*, in: A. Restivo, S. Ronchi Della Rocca and L. Roversi, editors, *Italian Conference on Theoretical Computer Science*, Lect. Notes in Comp. Sci. **2202** (2001), pp. 1–16.

[13] Gadducci, F. and G. V. Monreale, *A decentralized implementation of mobile ambients*, in: R. Heckel and G. Taentzer, editors, *Graph Transformation*, Lect. Notes in Comp. Sci. **forthcoming** (2008).

[14] Gadducci, F. and U. Montanari, *A concurrent graph semantics for mobile ambients*, in: S. Brookes and M. Mislove, editors, *Mathematical Foundations of Programming Semantics*, Electr. Notes in Theor. Comp. Sci. **45** (2001).

[15] Gadducci, F. and U. Montanari, *Observing reductions in nominal calculi* via *a graphical encoding of processes*, in: A. Middeldorp, V. van Oostrom, F. van Raamsdonk and R. de Vrijer, editors, *Processes, terms and cycles (Klop Festschrift)*, Lect. Notes in Comp. Sci. **3838** (2005), pp. 106–126.

[16] Gordon, A. D. and L. Cardelli, *Equational properties of mobile ambients*, Mathematical Structures in Computer Science **13** (2003), pp. 371–408.

[17] Leifer, J. and R. Milner, *Deriving bisimulation congruences for reactive systems*, in: C. Palamidessi, editor, *Concurrency Theory*, Lect. Notes in Comp. Sci. **1877** (2000), pp. 243–258.

[18] Merro, M. and F. Zappa Nardelli, *Behavioral theory for mobile ambients*, Journal of the ACM **52** (2005), pp. 961–1023.

[19] Milner, R., "Communication and Concurrency," Prentice Hall, 1989.

[20] Milner, R., "Communicating and Mobile Systems: the π-Calculus," Cambridge University Press, 1999.

[21] Milner, R., *Pure bigraphs: Structure and dynamics*, Information and Computation **204** (2006), pp. 60–122.

[22] Rathke, J. and P. Sobociński, *Deconstructing behavioural theories of mobility*, in: G. Ausiello, J. Karhumäki, G. Mauri and L. Ong, editors, *Theoretical Computer Science*, Lect. Notes in Comp. Sci. **forthcoming** (2008).

[23] Regev, A., E. Panina, W. Silverman, L. Cardelli and E. Shapiro, *Bioambients: an abstraction for biological compartments*, Theor. Comp. Sci. **325** (2004), pp. 141–167.

[24] Sobociński, P., "Deriving bisimulation congruences from reduction systems," Ph.D. thesis, BRICS, Department of Computer Science, University of Aarhus (2004).

[25] Sobociński, P. and J. Rathke, *Deriving structural labelled transitions for mobile ambients*, in: F. van Breugel and M. Chechik, editors, *Concurrency Theory*, Lect. Notes in Comp. Sci. **forthcoming** (2008).

Decidable Fragments of a Higher Order Calculus with Locations

Mikkel Bundgaard[1] Jens Chr. Godskesen[2]

Email: {mikkelbu, jcg}@itu.dk
The Programming, Logic, and Semantics group
IT University of Copenhagen

Bjørn Haagensen Hans Hüttel

Email: {bh, hans}@cs.aau.dk
Department of Computer Science
Aalborg University

Abstract

Homer is a higher order process calculus with locations. In this paper we study Homer in the setting of the semantic finite control property, which is a finite reachability criterion that implies decidability of barbed bisimilarity. We show that strong and weak barbed bisimilarity are undecidable for Homer. We then identify and compare two distinct subcalculi of Homer that both satisfy the semantic finite control property. One subcalculus is obtained by using a type system bounding the size of process terms. The other subcalculus is obtained by considering the image of the encoding of the finite control π-calculus in Homer.

Keywords: Decidability, higher order process passing, locations, semantic finite control

1 Introduction

The calculus Homer [5] is a higher order process calculus with nested location hierarchies and active process mobility. Its syntax and semantics are inspired by calculi such as Plain CHOCS [13] and the higher order π-calculus [11]. Similar to these calculi we have the ability to send a *passive* resource r (along the name a),

$$\overline{a}\langle r\rangle.p \,|\, a(x).q \longrightarrow p \,|\, q\{r/x\} \ .$$

Active process mobility and nested location hierarchies are introduced in the calculus by the location prefix, $a\langle r\rangle.p$, where r is an *active* resource computing at the

[1] Supported by grant no. 274-06-0415 and 2059-03-0031 from the Danish Research Council for Technology and Production and the IT University of Copenhagen (the CosmoBiz and BPL projects).

[2] Supported by grant no. 272-05-0258 from the Danish Research Agency.

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

location a. A process can take an active resource and bind it to a process variable using the complementary prefix $\overline{a}(x).q$ according to the following reduction rule

$$a\langle r\rangle.p \,|\, \overline{a}(x).q \longrightarrow p \,|\, q\{r/x\} \ .$$

We can communicate with processes residing in locations by allowing sequences of names in the prefixes. E.g. we can take the resource r from the location b inside the location a using the composite address ab

$$a\langle b\langle r\rangle \,|\, p'\rangle.p \,|\, \overline{ab}(x).q \longrightarrow a\langle p'\rangle.p \,|\, q\{r/x\} \ .$$

In a similar manner we can send a passive resource to a receiver residing in a sublocation.

$$a\langle b(x).p' \,|\, p''\rangle.p \,|\, \overline{ab}\langle r\rangle.q \longrightarrow a\langle p'\{r/x\} \,|\, p''\rangle.p \,|\, q \ .$$

Homer can encode persistent locations [5], mobility as in the Seal calculus [6], and name passing as in the π-calculus [1,2], thus exemplifying some of its expressive power. The extent to which higher order communication adds to the expressiveness of the first order π-calculus has been studied in [12], where it is shown that one can encode the higher order π-calculus, $HO\pi$, in the first order π-calculus. In the context of calculi with locations, the work on ambient calculi in [8] is related, albeit with a different perspective, as the subject of [8] is operators versus minimal Turing-complete fragments.

Intuitively, a *finite control* calculus is calculus where the control structure is finite. I.e. starting in any state, the number of states reachable via internal reduction steps is finite. In [3] the authors examine a finite control fragment of the ambient calculus. Similar to one of the approaches examined in this paper the finite control fragment is obtained by the usage of a type system instead of, as usual, relying on syntactic restrictions. Recent work in [7] considers a higher order calculus with few operators where it, perhaps surprisingly, holds that the calculus is Turing-complete but that barbed congruence is decidable.

The purpose of this work is to investigate decidability of barbed bisimilarity in Homer with all operators. Results of this type are useful as a basis for model- and equivalence-checking. Apart from the results mentioned above, few results of this type exist in the context of higher order calculi with locations, and the question is non-trivial since Homer can encode Turing machines. This paper shows that in a full higher order calculus with locations, finite control [4] is a complicated issue. In the context of CCS and the π-calculus [12] it has been shown that finite control can be obtained simply by prohibiting the use of the operator for parallel composition in recursive definitions [4]. The solution is not equally simple in higher order calculi such as Homer and $HO\pi$. There are several reasons for this. Firstly there is no explicit recursion or replication operator in Homer since recursion is a derived operator [6]. Moreover process-variables may be instantiated to arbitrary processes. But most important is the observation that even without using parallel composition in recursion, one can define a process with infinitely many non-barbed bisimilar reducts. We can construct such a process in Homer by using that process variables can occur at sublocations as in $a(x).\overline{a}\langle n\langle x\rangle\rangle$, where an extra level of nesting is added to the process received on channel a.

In order to find a decidable characterisation of a subcalculus of Homer for which barbed bisimilarity is decidable we explore two different approaches. The first approach is to use a type system which bounds the size of processes in terms of the number of parallel components, sequential length, and nesting of locations. The resulting subcalculus of Homer is called HFC_Γ. The resulting calculus is too restrictive and does not allow for infinite reductions. Therefore a recursion operator is added. Since processes in HFC_Γ can not acquire new free names, this ensures us that there are only finitely many different α-equivalence classes reachable from any process. The second approach is to consider an encoding of the π-calculus into Homer [1,2]. We apply it to the *finite control* π-calculus, $FC\pi$, and consider the image of the encoding as a subcalculus of Homer, HFC_π. It is shown that the finite control property is preserved by the encoding. HFC_π as well as HFC_Γ are subcalculi of Homer, and the finiteness results for HFC_π and HFC_Γ imply that the inclusions are strict. Moreover we prove that there are HFC_Γprocesses which do not have equivalent counterparts in HFC_π. First, the translation from $FC\pi$ to HFC_π results in processes which are not well typed in HFC_Γ. Second, any process in HFC_Γ with location nesting greater than some constant k does not have a counterpart in HFC_π.

2 The Calculus Homer

The syntax and semantics of Homer as presented by Bundgaard et. al. in [1] are given as follows. Let $\mathcal{N}$ be an infinite set of names and let $\mathcal{N}^*$ denote the set of all sequences of names formed by using names from $\mathcal{N}$, let $\mathcal{N}^+ \subset \mathcal{N}^*$ denote the set of non-empty sequences of names, and let $\tilde{n}$ range over finite sets of names. Let $a, b, n, m, \ldots$ range over $\mathcal{N}$, γ over $\mathcal{N}^*$ and δ over $\mathcal{N}^+$. Let $\mathcal{V}$ be an infinite set of process variables ranged over by $x, y, z, \ldots$. Finally let $\mathcal{U}$ be a set of recursion variables ranged over by X and Y. The set of Homer processes is given by the following grammar.

$$\begin{array}{l} p ::= \mathbf{0} \quad | \quad \mathrm{rec}\, X.p \quad | \quad p \,|\, p' \quad | \quad (n)p \quad | \quad \pi.p \quad | \quad x \quad | \quad X \\ \pi ::= \delta(x) \quad | \quad \overline{\delta}(x) \quad | \quad \overline{\delta}\langle p \rangle \quad | \quad \delta\langle p \rangle \end{array}$$

The primitives for the inactive process, recursion, parallel composition, and restriction have the same meaning as in other higher order process calculi. There are two prefixes representing a resource at a location δ, where δ is a sequence of names enabling addressing at sub-locations as described in the introduction. The active $\delta\langle p \rangle$, and the passive $\overline{\delta}\langle p \rangle$ prefix. The process p can perform internal reactions in $\delta\langle p \rangle$, and the context can communicate with p, this is not the case for p in $\overline{\delta}\langle p \rangle$. In Homer names are bound by restriction, $(n)p$, and process variables are bound by $\delta(x).p$ or $\overline{\delta}(x).p$, and recursion variables are bound as in $\mathrm{rec}\, X.p$. For a process p the set of free and bound names and variables are defined accordingly and denoted $\mathrm{fn}(p), \mathrm{bn}(p), \mathrm{fv}(p)$ and $\mathrm{bv}(p)$.

Let $\equiv_\alpha$ denote α-equivalence both with respect to names and variables. If $\mathrm{fv}(p) = \emptyset$, then p is called a *closed* process. Let $\mathcal{P}$ denote the set of processes given by the grammar (up to α-equivalence) and let $p, q, r, \ldots$ range over $\mathcal{P}$. Furthermore let $\mathcal{P}_c \subset \mathcal{P}$ denote the set of closed processes ranged over by the same meta variables as $\mathcal{P}$. Contexts are defined as process terms with a single hole.

Definition 2.1 (Contexts) *Homer* contexts $\mathscr{C}$ *and* evaluation contexts $\mathscr{E}$ *are given by the following grammars:*

$$\mathscr{C} ::= (-) \mid \operatorname{rec} X.\mathscr{C} \mid \mathscr{C} \mid p \mid (n)\mathscr{C} \mid \pi.\mathscr{C} \mid \delta\langle\mathscr{C}\rangle.p \mid \overline{\delta}\langle\mathscr{C}\rangle.p$$
$$\mathscr{E} ::= (-) \mid \mathscr{E} \mid p \mid (n)\mathscr{E} \mid \delta\langle\mathscr{E}\rangle.p'$$

Definition 2.2 (Well-formedness and linearity) *Let $p \in \mathcal{P}_c$. Then p is* linear *if for every sub-process $\delta(x).q$ or $\overline{\delta}(x).q$ of p, x occurs at most once in q. A process p is well-formed if it is linear and for every occurrence of $\operatorname{rec} X.p'$ in p, all occurrences of X in p' are guarded.*

In the following Homer^{wf} denotes the variant of Homer obtained by only using well-formed processes. Structural congruence is the least equivalence relation on $\equiv \subseteq \mathcal{P} \times \mathcal{P}$ which is closed under application of process contexts and which satisfies the following axioms.

$$p \mid \mathbf{0} \equiv p \qquad p \mid q \equiv q \mid p \qquad p \mid (q \mid r) \equiv (p \mid q) \mid r$$
$$(n)\mathbf{0} \equiv \mathbf{0} \qquad (n)p \mid q \equiv (n)(p \mid q), \text{ where } n \notin \operatorname{fn}(q) \qquad (n)(m)r \equiv (m)(n)r$$
$$\operatorname{rec} X.X \equiv \mathbf{0} \qquad \operatorname{rec} X.p \equiv p\{\operatorname{rec} X.p/X\}$$

As usual we also identify processes up to α-conversion. In order to handle addressing at sublocations the reduction rules are given using so-called path indexed contexts, $\mathscr{C}^{\tilde{m}}_{\gamma}$, where γ is the path to the hole, and $\tilde{m}$ the names bound in the hole.

Definition 2.3 (Path-indexed contexts) *Let $p, q \in \mathcal{P}_c$ and $\delta \in \mathcal{N}^+$ and $\gamma \in \mathcal{N}^*$. Then inductively define* path-indexed contexts *by*

$$\mathscr{C}^{\emptyset}_{\epsilon} \stackrel{def}{=} (-) \qquad \mathscr{C}^{\tilde{n}\tilde{m}}_{\delta\gamma} \stackrel{def}{=} \delta\langle(\tilde{n})\mathscr{C}^{\tilde{m}}_{\gamma} \mid p)\rangle.q, \text{ where } \tilde{n} \cap \gamma = \emptyset \ .$$

Vertical scope extrusion is defined using an open operator *on path contexts.*

$$\mathscr{C}^{\emptyset}_{\epsilon} \setminus \tilde{o} \stackrel{def}{=} \mathscr{C}^{\emptyset}_{\epsilon} \qquad \mathscr{C}^{\tilde{n}\tilde{m}}_{\delta\gamma} \setminus \tilde{o} \stackrel{def}{=} \delta\langle(\tilde{n} \setminus \tilde{o})\mathscr{C}^{\tilde{m}}_{\gamma} \setminus \tilde{o} \mid p\rangle.q,$$
$$\text{if } \mathscr{C}^{\tilde{n}\tilde{m}}_{\delta\gamma} = \delta\langle(\tilde{n})\mathscr{C}^{\tilde{m}}_{\gamma} \mid p\rangle.q \text{ and } (\tilde{m} \cup \tilde{n}) \cap \operatorname{fn}(\mathscr{C}^{\tilde{m}\tilde{n}}_{\delta\gamma}) = \emptyset \ .$$

When a resource is moved from a location it may be necessary to extend the scope of a name vertical through the location boundary using the open operator.

Definition 2.4 (Reduction relation) *The* reduction relation *is the least binary relation on $\mathcal{P}_c$ which is closed under structural congruence and evaluation contexts and which satisfies the following axioms.*

(Send) $\overline{\gamma\delta}\langle p\rangle.p' \mid \mathscr{C}^{\tilde{m}}_{\gamma}(\delta(x).q) \longrightarrow p' \mid \mathscr{C}^{\tilde{m}}_{\gamma}(q\{p/x\})$, *where* $\tilde{m} \cap (\operatorname{fn}(p) \cup \delta) = \emptyset$

(Take) $\mathscr{C}^{\tilde{m}}_{\gamma}(\delta\langle p\rangle.p') \mid \overline{\gamma\delta}(x).q \longrightarrow (\tilde{m} \cap \tilde{n})\big((\mathscr{C}^{\tilde{m}}_{\gamma} \setminus \tilde{n})(p') \mid q\{p/x\}\big)$
where $\tilde{n} = \operatorname{fn}(p), \tilde{m} \cap (\delta \cup \operatorname{fn}(q)) = \emptyset$

Note that in the (Take)-rule the names bound in the hole are vertically extruded if and only if they are actually free in p. For example in

$$a\langle(n)(b\langle r\rangle \mid p)\rangle \mid \overline{ab}(y).\,\mathrm{rec}\,X.(y \mid X) \ ,$$

the scope of n is extruded (through the location boundary of a) iff n is free in r, so each copy of r will share the name n. Otherwise r leaves the scope of n. A detailed discussion of this choice is presented in [6]. The rest of the side conditions in Definition 2.3 and Definition 2.4 are standard and prevent free names from being captured. Let $\longrightarrow^*$ denote the transitive and reflexive closure of $\longrightarrow$.

Definition 2.5 (Strong and weak barbs) *Define:*

- $p \downarrow n$ *if* $p \equiv (\tilde{a})(n\langle q\rangle.q' \mid q'')$, *where* $n \notin \tilde{a}$.
- $p \Downarrow n$ *if there is* p' *such that* $p \longrightarrow^* p'$ *and* $p' \downarrow n$.

Definition 2.6 (Strong and weak barbed bisimilarity) *A binary symmetric relation* $\mathcal{R} \subseteq \mathcal{P}_c \times \mathcal{P}_c$ *is called a* strong barbed bisimulation *if whenever* $(p,q) \in \mathcal{R}$ *the following holds:*

$$\textit{(i) If } p \downarrow n \textit{ then } q \downarrow n \quad \textit{(ii) If } p \longrightarrow p' \textit{ then } q \longrightarrow q' \textit{ and } (p',q') \in \mathcal{R} \ .$$

Processes p *and* q *are called strong barbed bisimilar, denoted* $p \sim q$, *if there is a strong barbed bisimulation* $\mathcal{R}$ *such that* $(p,q) \in \mathcal{R}$.

Weak barbed bisimilarity *is obtained by replacing (i) and (ii) with:*

$$\textit{(i) If } p \downarrow n \textit{ then } q \Downarrow n \quad \textit{(ii) If } p \longrightarrow p' \textit{ then } q \longrightarrow^* q' \textit{ and } (p',q') \in \mathcal{R} \ .$$

Processes p *and* q *are weakly bisimilar, denoted* $p \approx q$, *if there is a* weak bisimulation *such that* $(p,q) \in \mathcal{R}$.

For rec-free well-formed processes the following finiteness property can be proven by observing that all reductions strictly reduce the size of processes. The corresponding result does not hold for full Homer.

Lemma 2.7 *If* p *is a linear term built from the grammar without resorting to the* $\mathrm{rec}\,X.p$ *construction, then* $\{p' \mid p \longrightarrow^* p'\}/_{\equiv}$ *is finite.*

Strong, respectively weak, barbed bisimilarity are too coarse for many purposes, specifically in most cases only the congruence versions coincide with strong and weak bisimilarity. Indeed this is the case for Homer [6]. The coarsest equivalence is reduction bisimilarity, $\sim_r$. This holds since reduction bisimilarity only requires equivalent processes to match on τ-actions, whereas they, contrary to barbed bisimilarity, need not have equivalent barbs, i.e. observable actions. It is indeed necessary that $\sim_r$ is decidable for the congruences to be decidable. This is seen since decidability of reduction bisimilarity can be reduced to deciding barbed congruence, $\sim^c$. For any processes p and q let,

$$p \sim_r q \text{ if and only if } (\tilde{n})p \sim^c (\tilde{m})q, \text{ where } \tilde{n} = \mathrm{fn}(p) \text{ and } \tilde{m} = \mathrm{fn}(q) \ .$$

Therefore, even though our decidability results only applies to the non-congruence versions, they must indeed hold in any formalism for which the congruences are

decidable. For the positive results in this paper we will use the following finite control property.

Definition 2.8 (Semantic finite control (SFC)) *Let $\mathcal{A}$ be a process calculus and $\eqsim$ be some decidable equivalence such that $\eqsim \subseteq \approx, \sim$. Then $\mathcal{A}$ is called semantic finite control up to $\eqsim$ if the set $\{p' \mid p \longrightarrow^* p'\}/_{\eqsim}$ is finite.*

Proposition 2.9 *If $\mathcal{A}$ is SFC up to $\eqsim$, then $\approx$ and $\sim$ are decidable.*

As an indication of the expressiveness of SFC calculi, the next result shows that the traces of processes from calculi satisfying the SFC property are simple in structure.

Definition 2.10 (Barbed Trace) *Let $\mathcal{A}$ be a process calculus and let $p \in \mathcal{A}$ be a process. $\alpha = a_1 \cdots a_k$ is a* barbed trace *of p ending in p' if there exists*

$$p \longrightarrow^* \downarrow a_1 \longrightarrow \longrightarrow^* \downarrow a_2 \cdots \longrightarrow \longrightarrow^* p' \downarrow a_k \ .$$

We let $p \xrightarrow{\alpha}{}^* p'$ denote such a reduction sequence. The set of barbed traces generated by p is denoted $BTrace(p)$.

Lemma 2.11 (Pumping Lemma) *Let $\mathcal{A}$ be a process calculus which is SFC up to $\eqsim$ and assume $\eqsim$ respects barbed traces, and let $p \in \mathcal{A}$. Then there is a number n such that if $\alpha \in BTrace(p)$ and $|\alpha| \geq n$, then there are α_1, α_2, and α_3 such that $\alpha = \alpha_1\alpha_2\alpha_3$ and for each $i \in \mathbb{N}$ the following holds. (i) $\alpha_1\alpha_2^i\alpha_3 \in BTrace(P)$, (ii) $|\alpha_2| > 0$.*

Proof. Let $n = |\{p' \mid p \longrightarrow^* p'\}/_{\eqsim}| + 1$ and α a string, with $|\alpha| = n'$ and $n' \geq n$. Then there must exist a sequence of transitions visiting at least n' states. Thus some state must repeat. Denote the first of occurrence of this state p_j and the second p_k. Then $p \xrightarrow{\alpha_1}{}^* p_j \longrightarrow \xrightarrow{\alpha_2}{}^* p_k \longrightarrow \xrightarrow{\alpha_3}{}^* p'$, for some p'. By the pigeonhole principle one can now prove that $\alpha_1\alpha_2^i\alpha_3 \in BTrace(p)$ for all $i \in \mathbb{N}$. □

Corollary 2.12 *Let $\mathcal{A}$ be a process calculus which is SFC up to $\eqsim$ and assume that $\eqsim$ respects barbed traces. Then there is no $p \in \mathcal{A}$ and names a and b such that $\{a^ib^i\} = BTrace(p)$ for all $i \in \mathbb{N}$.*

In the following sections we first present some undecidability results, then we characterise SFC processes in two different ways: first using a type system which bounds the size of processes, and second using an encoding of the (finite control) π-calculus into Homer.

3 Undecidability Results

In [1,2] it is shown that Homer can encode the π-calculus. From that result it follows that Homer is Turing-complete. Although Turing-completeness usually implies that semantic properties of processes are undecidable, the recent paper [7] shows that undecidability of barbed congruence does not follow from the ability to encode Minsky machines in a termination preserving manner.

Definition 3.1 *A $\approx$-property $\mathcal{S}$ is a set of $\approx$-equivalence classes. $\mathcal{S}$ is* non-trivial *if there exists a $\mathcal{C}_1 \in \mathcal{S}$ and $\mathcal{C}_2 \notin \mathcal{S}$.*

Theorem 3.2 *If $\mathcal{S}$ is a non-trivial $\approx$-property, then $L_{\mathcal{S}} = \{p \mid \exists \mathcal{C} \in \mathcal{S}.\ p \in \mathcal{C}\}$ is undecidable.*

Proof. Reduction from the halting problem for Turing machines. Since $\mathcal{S}$ is non-trivial, there exist equivalence classes $\mathcal{C}_1 \in \mathcal{S}$ and $\mathcal{C}_2 \notin \mathcal{S}$. We choose a process $p_1 \in \mathcal{C}_1$. Further, we assume wlog that $p_2 \in \mathcal{C}_2$, where $p_2 = (\nu a)(\mathrm{rec}\, X.a(z).X \mid \mathrm{rec}\, Y.\overline{a}\langle \mathbf{0}\rangle.Y)$. Given a Turing machine M and an input x we can construct a Homer process $p_{M,x}$ whose only free name is w and such that w is only used to signal termination and such that $p_{M,x} \Downarrow w$ iff M halts on input x. Now construct the process $p_0 = (\nu w)(p_{M,x} \mid w.p_1)$. Then we have that $p_0 \in \mathcal{C}_1$ if M halts on x and that $p_0 \in \mathcal{C}_2$ if M does not halt on x. □

Corollary 3.3 *$\approx$ is undecidable.*

The proof of Theorem 3.2 remains valid also for barbed congruence and even for reduction bisimilarity. Consequently Corollary 3.3 also remains true for both equivalences. The analogous result for strong barbed bisimilarity is obtained by an encoding of the λ-calculus in Homer, inspired by the encoding in Plain CHOCS [13]. Assuming that a and $i \notin \mathrm{fn}([\![M]\!]) \cup \mathrm{fn}([\![N]\!])$.

$$[\![x]\!] \stackrel{\mathrm{def}}{=} x \qquad [\![\lambda x.M]\!] \stackrel{\mathrm{def}}{=} i(x).[\![M]\!] \qquad [\![MN]\!] \stackrel{\mathrm{def}}{=} (a)\left(a\langle [\![M]\!]\rangle \mid \overline{ai}\langle [\![N]\!]\rangle.\overline{a}(x).x\right) \ .$$

This encoding, while only being correct up to weak equivalence, is termination preserving, and moreover we know exactly how many steps the encoding needs to perform in order to simulate a reduction in a λ-term. Therefore

$$(i)([\![M]\!]) \sim \mathrm{rec}\, X.(a)(\overline{a} \mid a.X) \text{ iff } M \text{ diverges} \ ,$$

which is a reduction of the divergence problem for the λ-calculus.

4 The Calculus HFC_Γ

In this section we present the typed subcalculus HFC_Γ of Homer^{wf}. In the results that follow whenever a process is typed, the process is first transformed to a folded form which is now defined.

Definition 4.1 (Folding) *Let $X \in \mathrm{fv}(p)$ and define the binary relation* fold, $>$, *on process terms by the axioms*

$$p\{\mathrm{rec}\, X.p/X\} > \mathrm{rec}\, X.p \qquad p \mid \mathbf{0} > p \qquad \mathbf{0} \mid p > p \qquad (n)p > p \text{ if } n \notin \mathrm{fn}(p)$$
$$(n)(p \mid q) > (n)p \mid q \text{ if } n \notin \mathrm{fn}(q) \qquad (n)(p \mid q) > p \mid (n)q \text{ if } n \notin \mathrm{fn}(p)$$

and closure under the process contexts. p is said to be on folded form *if $p \not>$.*

Let $>^*$ denote the transitive closure of $>$. It is easy to see that any process is either on folded form, or can be brought on folded form. Moreover the relation $>$ is convergent and a sub-relation of $\equiv$. Therefore we can assume that all processes are on folded form for the remaining part of this section. The basic purpose of the type system is to ensure that the size of a typeable process is bounded. Types are

triples of natural numbers. In a type (d, w, s), d, w, and s are upper bounds for the depth of nested locations, width, i.e. the number of parallel components, and length of the prefix sequences.

Definition 4.2 (Types) Types *are triples* (d, w, s) *of non-negative natural numbers,* $\mathbb{N}$*, ranged over by* $S, T, \ldots$.

We write $(d, w, s) \leq (d', w', s')$ if $d \leq d', w \leq w'$, and $s \leq s'$.

Definition 4.3 (Type environments) *A* type environment *is a finite partial function* $\Gamma : \mathcal{N} \cup \mathcal{V} \hookrightarrow \mathbb{N} \times \mathbb{N} \times \mathbb{N}$.

Type environments can be regarded as finite sets of type assignments $a : T$, where $a \in \mathcal{N} \cup \mathcal{V}$ and $T \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ and an environment is written $\{a_1 : T_1, \ldots, a_n : T_n\}$ where $a_i \neq a_j$ when $i \neq j$. Type environments can be extended. This is written $\Gamma \cup \{a : T\}$, and is only defined if a is not defined in Γ. Below we let φ range over δ and $\overline{\delta}$. There are two type judgement relations for HFC_Γ.

Definition 4.4 *The* type relation for names, $\vdash_n$*, is given by the following rules*

$$(\textsc{TName}) \qquad \Gamma \cup \{a : (d, w, s)\} \vdash_n a : (d, w, s)$$

$$(\textsc{TSeqName}) \quad \frac{\Gamma \vdash_n \delta : (d, w, s) \qquad \Gamma \vdash_n a : (d', w', s')}{\Gamma \vdash_n \delta a : (d', w', s')}, \quad \textit{if } (d, w, s) \leq (d', w', s')$$

The type relation for processes *on folded form,* $\vdash$*, is given by the following rules*

$$(\textsc{TProcVar}) \qquad \Gamma \cup \{x : (d, w, s)\} \vdash x : (d, w, s)$$

$$(\textsc{TRecVar}) \qquad \Gamma \cup \{X : (d, w, s)\} \vdash X : (d, w, s)$$

$$(\textsc{TNil}) \qquad \Gamma \vdash \mathbf{0} : (0, 1, 0)$$

$$(\textsc{TNew}) \quad \frac{\Gamma \cup \{n : (d', w,' s')\} \vdash p : (d, w, s)}{\Gamma \vdash (n)p : (d, w, s)}$$

$$(\textsc{TPar}) \quad \frac{\Gamma \vdash p : (d, w, s) \qquad \Gamma \vdash q : (d', w', s')}{\Gamma \vdash p \,|\, q : (\max\{d, d'\}, w + w', \max\{s, s'\})}$$

$$(\textsc{TTake}|\textsc{TIn}) \quad \frac{\Gamma \cup \{x : (d, w, s)\} \vdash p : (d', w', s') \qquad \Gamma \vdash_n \varphi : (d'', w'', s'')}{\Gamma \vdash \varphi(x).p : (d', w', 1 + s')}, \quad \textit{if } d \geq d''$$

$$(\textsc{TSend}|\textsc{TOut}) \quad \frac{\begin{array}{c}\Gamma \vdash_n \varphi : (d'', w'', s'') \\ \Gamma \vdash p : (d', w', s') \qquad \Gamma \vdash q : (d, w, s)\end{array}}{\Gamma \vdash \varphi\langle p \rangle.q : (\max\{d' + 1, d\}, w' + w, \max\{s', s + 1\})}, \quad \textit{if } d' \leq d''$$

$$(\textsc{TRec}) \quad \frac{\Gamma \cup \{X : (d', w', s')\} \vdash p : (d, w, s)}{\Gamma \vdash \mathrm{rec}\, X.p : (d', w', s')}, \quad \textit{if } d \leq d', w \leq w'$$

The rules are fairly self explanatory. It is important that processes are on folded form when they are typed. Otherwise the type would not be well-defined since one can use $\equiv$ to unfold recursion and to add $\mathbf{0}$-processes. The (TRec) rule enforces, through its side conditions, that the recursion variable cannot be placed freely in p. Specifically in $\mathrm{rec}\, X.p$, if the recursion variable occurs free in p, then there cannot be any occurrences of $\mid$ in p. This is similar to the finite control condition in $FC\pi$. There are no restrictions on the sequence component of the types.

Definition 4.5 (Well-typedness) *A process p is well-typed in Γ if there is some (d, w, s) such that $\Gamma \vdash p : (d, w, s)$. A process p is* well typed *if there is some Γ such that p is well typed in Γ.*

In the following, the notation is overloaded so $\vdash$ denotes $\vdash$ as well as $\vdash_n$ relying on the context to make it clear which one is meant. With this type system we can show that the number of different well-typed processes reachable starting from any well-typed process is finite up to $\equiv$.

Theorem 4.6 (Subject reduction) *If $\Gamma \vdash p : (d, w, s)$, then there is some k such that for all $p \longrightarrow p'$, $\Gamma \vdash p' : (d', w', s')$ for some d', w', and s' such that $d' \leq d$, $w' \leq w$, and $s' \leq k$.*

It is necessary to allow for the type of the third component to grow due to unfolding of recursion. But the size of the third component can be uniformly bounded in one step reductions as well as reductions of length greater than one.

Corollary 4.7 *If $\Gamma \vdash p : (d, w, s)$ then if $p \longrightarrow^* p'$ for some p', then there is (d', w', k) such that $\Gamma \vdash p' : (d', w', s')$ and $d' \leq d, w' \leq w$, and $s' \leq k$.*

Lemma 4.8 *Let $p \longrightarrow^* p'$. Then $\mathrm{fn}(p') \subseteq \mathrm{fn}(p)$.*

Proposition 4.9 *Let $A \subset \mathcal{N}$ be a finite set of names. Then for all natural numbers d', w', s', there are only finitely many α non-equivalent processes p such that*

- *p is on folded form*
- *$\Gamma \vdash p : (d, w, s)$, where $d \leq d', w \leq w'$, and $s \leq s'$.*
- *$\mathrm{fn}(p) \cup \mathrm{bn}(p) \subseteq A$*

Proposition 4.10 *The set of reachable configurations is finite*

$$|\{p' \mid \Gamma \vdash p : (d, w, s), \text{ and } p \longrightarrow^* p', \text{ and } \Gamma \vdash p' : (d', w', s')\}| < \infty \ .$$

A process on folded form may be thought of as a representative of an $\equiv$-equivalence class. Proposition 4.10 then says that there are only finitely many reachable processes up to $\equiv$.

Lemma 4.11 *Assume $\Gamma \vdash p : T$. Then $p \downarrow_n$ and $p \Downarrow_n$ are decidable.*

Theorem 4.12 *Strong and weak barbed bisimilarity are decidable for HFC_Γ.*

We now show that within the current setting the types bounding the depth, width, and length of prefix sequences are necessary to obtain finite control. In the following let HFC_Γ^{-d}, HFC_Γ^{-w}, and HFC_Γ^{-s} denote subcalculi of Homer^{wf} defined in the same way as HFC_Γ, but with the type-system slightly modified by removing the i'th component of the types and adapting the rules accordingly. The corresponding

typing judgements are $\Gamma^{-d} \vdash p$, $\Gamma^{-w} \vdash p$, and $\Gamma^{-s} \vdash p$ respectively.

Proposition 4.13 *Strong/weak barbed bisimilarity are undecidable for* HFC_{Γ}^{-d} .

Proof. [Sketch] The proposition is proven by showing that HFC_{Γ}^{-d} can encode Minsky machines [10]. A Minsky machine consists of a set of instructions $\{L_1, \ldots L_k\}$ where the instructions operates on two counters c_1 and c_2. For all instructions L_i that loop, we add a replica L'_i of L_i to the instruction set and encode the modified instruction set, $\{L_1, \ldots L_k, L'_i \ldots, L'_j\}$. This does not change the semantics of M. Each instruction L_i is either $\mathrm{Inc}(c_j, n)$, which increments the value of counter c_j and jumps to the next instruction n, or $\mathrm{Dec}(c_j, n, m)$ which jumps to instruction n if $c_j = 0$, otherwise the counter c_j is decremented by 1 followed by a jump to instruction m. A *program counter* (PC) keeps track of the currently executing instruction. Execution starts with the first instruction and halts if the PC gets assigned a value outside the range $1, \ldots, k$. The semantics of a Minsky machine is a transition system over configurations (i, c_1, c_2), where i is the PC and c_i, the values of the counters generated by the rules.

$$(\textsc{Inc}:) \quad \frac{i = \mathrm{Inc}(c_j, n) \quad c'_j = c_j + 1 \quad c'_{j-1} = c_{j-1}}{(i, c_1, c_2) \longrightarrow (n, c'_1, c'_2)}$$

$$(\textsc{Dec-1}:) \quad \frac{i = Dec(c_j, n, m) \quad c_j = 0}{(i, c_1, c_2) \longrightarrow (m, c_1, c_2)}$$

$$(\textsc{Dec-2}:) \quad \frac{i = Dec(c_j, k) \quad c_j \neq 0 \quad c'_j = c_j - 1 \quad c'_{j-1} = c_{j-1}}{(i, c_1, c_2) \longrightarrow (i + 1, c'_1, c'_2)}$$

In the following we write $a\langle \mathbf{0} \rangle.\mathbf{0}$ as a and omit trailing occurrences of $\mathbf{0}$ in e.g. $\delta\langle p \rangle.\mathbf{0}$. Numbers are encoded as $[\![0]\!] = z$ and $[\![n+1]\!] = n\langle [\![n]\!] \rangle$. In the encoding we use two special register locations, r_1 and r_2 from which the values of the counters c_1 and c_2 are read and saved. Instructions are encoded as follows.

$$[\![\mathrm{Inc}(c_i, n)]\!] = \mathrm{rec}\, X.l_m.\, \mathrm{Add}(c_i, n), \text{ where } \mathrm{Add}(c_i, n) = \overline{r_i}(x).r_i\langle n\langle x \rangle\rangle.\overline{l_n}.X \ .$$

For the encoding of $\mathrm{Dec}(c_i, n, m)$ we split the encoding into several parts.

$$\mathrm{Get}(c_i) = \mathrm{rec}\, Y.l_m.\, \mathrm{Get1}(c_i), \text{ where } \quad \mathrm{Get1}(c_i) = \overline{r_i}(x).a\langle x \rangle.b.Y$$

$$Zero(n) = \mathrm{rec}\, X'.\overline{az}(y).r_i\langle z\langle y \rangle\rangle.\overline{l_n}.\overline{b}.X' \quad NonZero(m) = \mathrm{rec}\, Y'.\overline{an}(y).r_i\langle y \rangle.\overline{l_0}.\overline{b}.Y'$$

Now the encoding of the if-then-else instruction is given as follows

$$[\![\mathrm{Dec}(c_i, n, m)]\!] = (a, b)(\mathrm{Get}(c_i) \mid \mathrm{Zero}(n) \mid \mathrm{NonZero}(m)) \ .$$

Let m be the number of indices in the modified instruction set. The full encoding is defined by encoding the instructions in parallel with the encoding of the counters.

$$\prod_{i \in \{1,\ldots,m\} \setminus \{k\}} [\![L_i]\!] \mid r_1\langle [\![c_1]\!] \rangle \mid r_2\langle [\![c_2]\!] \rangle$$

□

For HFC_Γ^{-w} and HFC_Γ^{-s} we have the weaker result.

Lemma 4.14 *The calculi HFC_Γ^{-w} and HFC_Γ^{-s} are not finite control.*

Proof. The two cases are established by providing counter-examples.
Case HFC_Γ^{-w}: $\overline{a}\langle(n)n\langle\mathbf{0}\rangle.\mathbf{0}\rangle.\mathbf{0} \mid \mathrm{rec}\, X.a(x).\overline{a}\langle x\rangle.X \mid \mathrm{rec}\, Y.a(x).\overline{a}\langle n\langle\mathbf{0}\rangle \mid x\rangle.Y$
Case HFC_Γ^{-s}: $\overline{a}\langle(n)n\langle\mathbf{0}\rangle.\mathbf{0}\rangle.\mathbf{0} \mid \mathrm{rec}\, X.a(x).\overline{a}\langle x\rangle.X \mid \mathrm{rec}\, Y.a(x).\overline{a}\langle(n')n'\langle\mathbf{0}\rangle.x\rangle.Y$ □

Recalling the comments in Sec. 2, Theorem 4.12 is, in our setting, an upper bound on the expressivity of a calculus for which strong and weak barbed congruence can be decidable. We believe this is also the case for HFC_Γ^{-w} and HFC_Γ^{-s}, but have not been able to improve on the result of Lemma 4.14.

5 The Calculus HFC_π

Contrast to Homer, the π-calculus is a first order calculus without a primitive notion of locations. The syntax and main reduction rule is reminiscent of Homer. However, whereas processes are passed over named channels in Homer, only names can be passed in the π-calculus. We briefly present the π-calculus and recommend [12,9] for details.

$$P ::= \mathbf{0} \quad | \quad P \mid Q \quad | \quad (\nu n)P \quad | \quad \mathrm{rec}\, X.P \quad | \quad X \quad | \quad \overline{n}\langle m\rangle.P \quad | \quad n(m).P$$

The finite control segment is obtained by imposing the following simple restrictions on the recursion operator, $\mathrm{rec}\, X.P$. First all occurrences of X in P must occur under a prefix, $\overline{n}\langle m\rangle$ or $n(m)$. Secondly there should be no parallel compositions in P. These two conditions are sufficient for obtaining finite control in the π-calculus [4]. Process contexts and evaluation contexts can be defined by disregarding all cases mentioning locations, denoting processes by capital instead of lower case letters, and writing (νn) for (n) in Definition 2.1. Structural congruence $\equiv_\pi$ is obtained in a similar manner. The semantics of the finite control π-calculus is given as the least binary relation $\longrightarrow_\pi$ over π-calculus terms closed under evaluation contexts and $\equiv_\pi$ and satisfying the following axiom

$$(\textsc{React})\ \frac{}{n(m).P \mid \overline{n}\langle m'\rangle.Q \longrightarrow_\pi P\{m'/m\} \mid Q}\ .$$

Definition 5.1 (Strong and weak barbed bisimilarity) *We define* strong *and* weak barbs *as usual:*

- *Assuming that $n \notin \tilde{a}$ we have $P \downarrow_\pi \overline{n}$ if $P \equiv (\nu\tilde{a})(\overline{n}\langle m\rangle.Q \mid Q')$ and $P \downarrow_\pi n$ if $P \equiv (\nu\tilde{a})(n(m).Q \mid Q')$.*
- *$P \Downarrow_\pi n$ if there is P' such that $P \longrightarrow_\pi^* P'$ and $P' \downarrow_\pi n$.*

A binary symmetric relation $\mathcal{R}$ over the set of π-calculus terms is called a strong barbed bisimulation *if whenever $(P,Q) \in \mathcal{R}$ the following holds:*

(i) If $P \downarrow_\pi n$ then $Q \downarrow_\pi n$ (ii) If $P \longrightarrow_\pi P'$ then $Q \longrightarrow_\pi Q'$ and $(P',Q') \in \mathcal{R}$.

Processes P and Q are called strong barbed bisimilar, denoted $P \sim_\pi Q$, if there is a strong barbed bisimulation $\mathcal{R}$ such that $(P,Q) \in \mathcal{R}$. Weak barbed bisimulation, *denoted $\approx_\pi$, is defined by modifying (i) and (ii) in the same way as in Definition 2.6.*

Next follows a brief account of the encoding of the π-calculus in Homer from [2] applied to $FC\pi$. The full encoding, $[\![\cdot]\!]_2$, is defined in terms of an encoding of names, $[\![\cdot]\!]$, and an encoding of processes, $[\![\cdot]\!]_1$. A π-calculus name n is encoded as a mobile resource $[\![n]\!]$ that performs two tasks; sending and receiving.

$$\begin{aligned} Send_n &\stackrel{\text{def}}{=} v(x).c(y).\overline{n}\langle x\rangle.y \\ Receive_n &\stackrel{\text{def}}{=} c(x).n(y).(a)\big(a\langle x\rangle \mid \overline{ab}\langle y\rangle.\overline{a}(z).z\big) \\ [\![n]\!] &\stackrel{\text{def}}{=} s\langle Send_n\rangle \mid r\langle Receive_n\rangle \end{aligned}$$

$Send_n$ expects the encoding of the name to be communicated on v, and the continuation of the prefix on c. $Receive_n$ expects the encoding of the continuation and is then ready to synchronise with the resulting $Send_n$ prefix. The significant cases of the encoding are input, output, and restriction.

$$\begin{aligned} [\![\overline{n}\langle m\rangle.P]\!]_1 &\stackrel{\text{def}}{=} (a)\big(a\langle n'\rangle \mid \overline{asv}\langle m'\rangle.\overline{asv}\langle [\![P]\!]_1\rangle.\overline{as}(z).\overline{a}(z').z\big) \\ [\![n(x).P]\!]_1 &\stackrel{\text{def}}{=} (a)\big(a\langle n'\rangle \mid \overline{arc}\langle b(x).\,[\![P]\!]_1\rangle.\overline{ar}(z).\overline{a}(z').z\big) \\ [\![(\nu n)P]\!]_1 &\stackrel{\text{def}}{=} (n)\big([\![P]\!]_1\,\{[\![n]\!]/n'\}\big) \end{aligned}$$

Note that the names n' and m' are free process variables which will be replaced by $[\![n]\!]$ and $[\![m]\!]$ on top-level in the encoding. The encoding of $\overline{n}\langle m\rangle.P$ sends $[\![m]\!]$ to the $Send_n$ process followed by the encoding of P. The $Send_n$-process is now located in a and ready to send $[\![m]\!]$ on n after which it becomes $[\![P]\!]_1$. This $Send_n$ process is now fetched from a and placed on top-level ready to communicate with $Receive_n$. The encoding of an input $n(x).P$ sends the encoding of the continuation prefixed with an input on which it can receive the $[\![m]\!]$ which was sent by $Send_n$. The actual π-calculus communication can now be executed before the result is finally fetched from a and placed at the top level. The $a(z')$ in both encodings garbage collects the unused part of the encoding of a name. It is assumed that there is a one-to-one mapping between π-calculus names n and process variables n'. The encoding is homomorphic on $\mathbf{0}$, $\mid$, and $\mathrm{rec}\, X.P$. The full encoding of a π-calculus process P with free names $n_1,\ldots,n_m$ is $[\![P]\!]_2 \stackrel{\text{def}}{=} [\![P]\!]_1\,\{[\![n_1]\!]/n'_1,\ldots[\![n_m]\!]/n'_m\}$, where $n'_1,\ldots n'_m$ are names in bijection with $n_1,\ldots,n_m$.

Example 5.2 The encoding of $P = \overline{n}\langle m\rangle \mid n(x).\overline{x}\langle x\rangle \longrightarrow_\pi \overline{m}\langle m\rangle$.

$$\begin{aligned} [\![P]\!]_2 &= \Big[(a)\big(a\langle n'\rangle \mid \overline{asv}\langle m'\rangle.\overline{asv}\langle [\![\mathbf{0}]\!]_1\rangle.\overline{as}(z).\overline{a}(z').z\big) \mid \\ &\qquad (a)\big(a\langle n'\rangle \mid \overline{arc}\langle b(x).\,[\![\overline{x}\langle x\rangle]\!]_1\rangle.\overline{ar}(z).\overline{a}(z').z\big)\Big]\{[\![n]\!]/n', [\![m]\!]/m'\} \\ &= (a)\big(a\langle [\![n]\!]\rangle \mid \overline{asv}\langle [\![m]\!]\rangle.\overline{asv}\langle [\![\mathbf{0}]\!]_1\rangle.\overline{as}(z).\overline{a}(z').z\big) \mid \\ &\qquad (a)\big(a\langle [\![n]\!]\rangle \mid \overline{arc}\langle b(x).\,[\![\overline{x}\langle x\rangle]\!]_1\rangle.\overline{ar}(z).\overline{a}(z').z\big) \end{aligned}$$

Thus we have the reductions

$$\begin{aligned}\llbracket P\rrbracket_2 &\longrightarrow^* \overline{n}\langle\llbracket m\rrbracket\rangle \mid n(y).(a)\big(a\langle b(x).\,\llbracket\overline{x}\langle x\rangle\rrbracket_1\rangle \mid \overline{ab}\langle y\rangle.\overline{a}(z).z\big)\\ &\longrightarrow^* (a)\big(a\langle b(x).\,\llbracket\overline{x}\langle x\rangle\rrbracket_1\rangle \mid \overline{ab}\langle\llbracket m\rrbracket\rangle.\overline{a}(z).z\big)\\ &\longrightarrow^* \llbracket\overline{x}\langle x\rangle\rrbracket_1\,\{\llbracket m\rrbracket/x\} = \llbracket\overline{m}\langle m\rangle\rrbracket_2\end{aligned}$$

Theorem 5.3 (Dynamic correspondence [2]) *$P \longrightarrow_\pi P'$ iff $\llbracket P\rrbracket_2 \longrightarrow^{10} \llbracket P'\rrbracket_2$.*

Thus if there are only finitely many reducts in the π-calculus, this must be reflected in the encoded process.

Corollary 5.4 *$|\{P' \mid P \longrightarrow_\pi^* P'\}| < \infty$ implies $|\{\llbracket P''\rrbracket \mid \llbracket P\rrbracket \longrightarrow^* \llbracket P''\rrbracket\}|/_\equiv < \infty$.*

The next proposition generalises the preceding lemma to arbitrary reductions in the encoded process.

Proposition 5.5 *$|\{P' \mid P \longrightarrow_\pi^* P'\}| < \infty$ implies $|\{c \mid \llbracket P\rrbracket \longrightarrow^* c\}|/_\equiv < \infty$.*

Corollary 5.6 *If P is a $FC\pi$-process. Then $P \downarrow_\pi n$ and $P \Downarrow_\pi n$ are decidable.*

Let HFC_π denote the subset of Homer-processes obtained as the taking the encoding of all $FC\pi$ processes together with their reducts.

Theorem 5.7 *HFC_π is SFC up to to $\equiv$.*

Although Theorem 5.7 only gives us SFC up to $\equiv$, a stronger statement can be obtained by using the labelled transition semantics without $\equiv$ instead [5].

6 Comparing Homer, Homer^{wf}, HFC_Γ, and HFC_π

In this section we show that Homer, HFC_Γ, and HFC_π are different calculi with respect to weak bisimilarity. Let $A, B \in \{Homer, \text{Homer}^{wf}, HFC_\Gamma, HFC_\pi\}$. We compare the calculi according to the following criteria.

$$A \lessapprox B \text{ if for all } p \in A \text{ there exists } q \in B \text{ such that } p \approx q$$

$$A \approx B \text{ if } A \lessapprox B \text{ and } B \lessapprox A. \qquad A \not\approx B \text{ if } A \not\lessapprox B \text{ and } B \not\lessapprox A\ .$$

Proposition 6.1 *(i) $HFC_\Gamma \lessapprox Homer^{wf}$ and (ii) $Homer^{wf} \not\lessapprox HFC_\Gamma$*

Proof. (i) holds since any HFC_Γ process is also a Homer^{wf}-process. (ii) holds since one can easily construct a Homer^{wf}-process which has an infinite sequence of reductions going through mutually non-equivalent states. This is not possible in HFC_Γ due to Proposition 4.10. □

In a similar manner we get.

Proposition 6.2 *(i) $HFC_\pi \lessapprox Homer$ and (ii) $Homer \not\lessapprox HFC_\pi$*

The next question to be answered is whether HFC_Γ is equivalent to HFC_π. First observe that membership in the two calculi is indeed a decidable problem.

Proposition 6.3 *Let $p \in Homer$. Then the following are decidable. (i) $p \in HFC_\Gamma$ and (ii) $p \in HFC_\pi$*

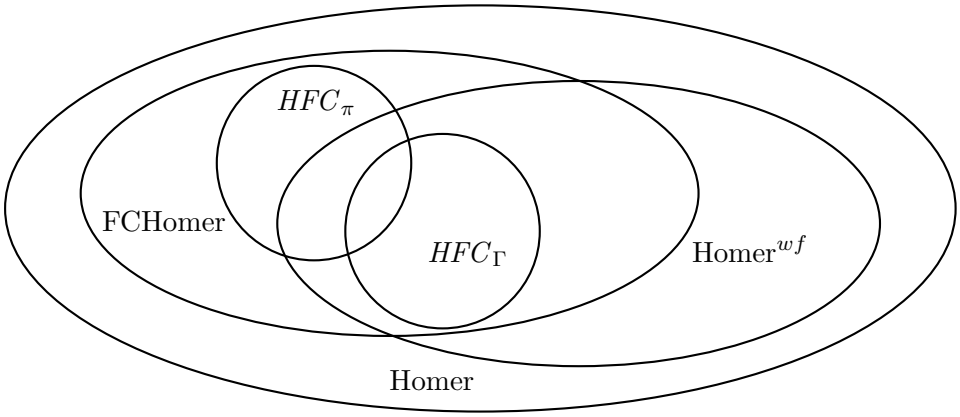


Fig. 1. Relationship between Homer, FCHomer, Homer^{wf}, HFC_Γ, and HFC_π.

Proposition 6.4 *We have* $HFC_\Gamma \not\lessapprox HFC_\pi$.

Letting FCHomer denote the full subcalculus of Homer where barbed bisimilarity is decidable we depict the calculi and inclusions with respect to $\approx$ in Figure 1. In Figure 1 the inclusion of HFC_Γ in Homer^{wf} and the inclusion of HFC_π in Homer are strict with respect to $\approx$. Moreover we conjecture that that $HFC_\pi \not\lessapprox HFC_\Gamma$. Obviously they have a non-empty intersection since e.g. the $\mathbf{0}$-process is typeable as well as the encoding of $\mathbf{0}$. We also note that HFC_Γ and HFC_π are not closed with respect to $\approx$. This can be seen since $\mathbf{0}$ is both an encoding of a π-calculus process, and a typeable process.

7 Conclusion

This paper deals with decidability of barbed bisimilarity in a higher order process calculus with locations called Homer. The problem seems much more complicated than for CCS and the π-calculus. Two subcalculi of Homer are characterised where bisimilarity is decidable in both of those calculi. To fix a point of reference we defined semantic finite control up to a decidable relation $\doteq$. Semantic finite control then implies decidability of any relation containing $\doteq$. However since Homer is Turing-complete most semantic properties are undecidable. In particular barbs and hence barbed bisimilarity.

These results lead us to pursue alternative characterisations. One is to define a type system which bounds the size of processes. The typed calculus is a subcalculus of Homer which is semantic finite control. The other characterisation draws on results from the finite control π-calculus and a relatively recent published encoding of the π-calculus into Homer. Combining these results we again obtain a subcalculus of Homer which is semantic finite control.

In regards to future work the most pressing issue is whether our results extends to congruences. In Homer, early context bisimilarity characterises barbed congruence. We have not, so far, succeeded in extending the present results to barbed congruence, or equivalently early context bisimilarity. The reason for this is that in the presence of higher order communication the early (labelled) context bisimilar-

ity relation does not get rid of the universal quantification over contexts. However recent work in [7] shows that in a certain case it is possible to derive a quantification free characterisation of barbed congruence. Whether the same approach is applicable in our setting would be interesting to know.

We have shown that $HFC_\Gamma \not\lesssim HFC_\pi$ and that Homer is strictly more expressive than both of these calculi. It is also clear that at the syntactic level, there are processes in HFC_π which are not in HFC_Γ, e.g. processes which are the encoding of π-calculus processes for which an input bound name occurs free more than once behind the prefix. We conjecture that also $HFC_\pi \not\lesssim HFC_\Gamma$. However we note that if the conjecture does not holds, then it is indeed possible that HFC_π could be embedded in HFC_Γ. Thus showing that despite the rather strict conditions imposed by the type-system, HFC_Γ would be at least as expressive as the finite control π-calculus. Finally HFC_π is at least as expressive as the finite control π-calculus. It would be of interest to know more about the expressive power of HFC_Γ and HFC_π. Also of interest is to find some more general notion of what decidable procedures could characterise semantic finite control. A natural extension of such work would then be to study the relationship between various notions of semantic finite control.

References

[1] Bundgaard, M., T. Hildebrandt and J. C. Godskesen, *A CPS encoding of name-passing in higher-order mobile embedded resources*, Theoretical Computer Science **356** (2006), pp. 422–439.

[2] Bundgaard, M., T. Hildebrandt and J. C. Godskesen, *On encoding the π-calculus in higher-order calculi*, Technical Report TR-2008-106, IT University of Copenhagen (2008).

[3] Charatonik, W., A. D. Gordon and J.-M. Talbot, *Finite-control mobile ambients*, in: *Proceedings of ESOP'02*, LNCS **2305** (2002), pp. 295–313.

[4] Dam, M., *On the decidability of process equivalences for the π-calculus*, Theoretical Computer Science **183** (1997), pp. 215–228.

[5] Godskesen, J. C. and T. Hildebrandt, *Extending Howe's method to early bisimulations for typed mobile embedded resources with local names*, in: *Proceedings of FSTTCS'05*, LNCS **3821** (2005), pp. 140–151.

[6] Hildebrandt, T., J. C. Godskesen and M. Bundgaard, *Bisimulation congruences for Homer — a calculus of higher order mobile embedded resources*, Technical Report TR-2004-52, IT University of Copenhagen (2004).

[7] Lanese, I., J. A. Pérez, D. Sangiorgi and A. Schmitt, *On the expressiveness and decidability of higher-order process calculi*, in: *Proceedings of LICS'08* (2008), pp. 145–155.

[8] Maffeis, S. and I. Phillips, *On the computational strength of pure ambient calculi*, Theoretical Computer Science **330** (2005), pp. 501–551.

[9] Milner, R., "Communicating and Mobile Systems: the π-calculus," Cambridge University Press, 1999.

[10] Minsky, M., "Computation: Finite and infinite machines," Prentice-Hall, 1967.

[11] Sangiorgi, D., "Expressing mobility in Process Algebras: First-Order and Higher-Order Paradigms," Ph.D. thesis, University of Edinburgh, Dept. of Computer Science (1993).

[12] Sangiorgi, D. and D. Walker, "The π-Calculus: a Theory of Mobile Processes," Cambridge University Press, 2001.

[13] Thomsen, B., *Plain CHOCS: A second generation calculus for higher order processes*, Acta Informatica **30** (1993), pp. 1–59.

Hard life with weak binders

Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari

Dipartimento di Informatica, Università di Pisa, Italy

Roberto Zunino

Dipartimento di Informatica e Telecomunicazioni, Università di Trento, Italy

Abstract

We introduce *weak binders*, a lightweight construct to deal with fresh names in nominal calculi. Weak binders do not define the scope of names as precisely as the standard ν-binders, yet they enjoy strong semantic properties. We provide them with a denotational semantics, an equational theory, and a trace inclusion preorder. Furthermore, we present a trace-preserving mapping between weak binders and ν-binders.

Keywords: Nominal calculi, variable binding, alpha-conversion, freshness

1 Introduction

Over the last few years *naming* has been envisaged as a suitable abstraction for capturing and handling a variety of computational concepts, like distributed objects, cryptographic keys, session identifiers. Also, the dynamicity issues usually arising in distributed computing (e.g., network reconfiguration, module versioning) may be usefully explained in terms of naming disciplines such as fresh name generation, binding and scoping rules. The π-calculus [12,18] is probably the most illustrative example of nominal calculi, in which many of the concepts outlined above have been formally modelled and explained. Nominal calculi manipulate names via explicit binders that define their scope. The standard example is the π-calculus restriction operator νn. A ν-binder also declares that a fresh name has to be created. A broad variety of formal theories [8,9,20,17,13,14,4] developed in the last few years shows the intrinsic difficulties of handling naming and freshness.

This paper aims at contributing to this line of research. Our motivating starting point is to understand what is the actual gain in using ν-binders to deal with fresh names. Indeed, the equational theory of ν-binders allows for freely moving them

⋆ This research has been partially supported by EU-FETPI Global Computing Project IST-2005-16004 SENSORIA (Software Engineering for Service-Oriented Overlay Computers)

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

almost anywhere in a process (except escaping from a recursion). So, one might wonder whether ν-binders can be omitted in a process, and replaced by a more primitive construct, e.g. an atomic action to be interpreted as a *gensym()* that explicitly creates a fresh name.

We introduce a nominal calculus with *weak binders*, a construct for generating fresh names as an atomic action, without explicit ν-binders. Our calculus slightly extends Bergstra and Klop's Basic Process Algebras [3], by allowing parametrized atomic actions $\alpha(r)$, that abstract from dispatching the action α to the object r. Objects can be freshly created through the special action $new(n)$, our "weak binder".

We study under which conditions a weakly bound process can be treated coherently with a process with ν-binders. For instance, in the weakly bound process $p = new(n) \cdot \alpha(n) + new(m) \cdot \alpha(m)$ there is no confusion between the scopes of the "bound" names n and m, and so p is equivalent to the "strongly bound" process $P = \nu n.\nu m.(new(n) \cdot \alpha(n) + new(m) \cdot \alpha(m))$. We shall then say that p is *well-bound*, and that P is its *bindification*. This transformation makes precise the scopes of names in weakly bound processes, by inserting the ν-binders at the right points. This is not always possible, however, e.g. in the process $new(n) \cdot (\varepsilon + new(n)) \cdot \alpha(n)$ there is an inherent ambiguity, because we cannot tell whether the action α has to be done on the object created by the first or by the second *new*. When bindification is possible, we prove that the semantics of the weakly bound and of the bindified processes are trace equivalent.

A further contribution is a trace inclusion preorder $\precsim$ for weakly bound processes: when $p \precsim q$, the traces of p are included in those of q. We compare this preorder with a trace inclusion preorder for strongly bound processes. Preorders of processes are a relevant and non-trivial aspect of subtyping/subeffecting for type and effect systems [1]. Also, thay can be udes to study the compliance of contracts with implementations and subcontract relations in calculi for Web services [5,6].

We envisage the impact of our approach as follows. Our main result is the formal definition of a methodology for handling the freshness of names without resorting to explicit binders. The overall outcome of our semantical investigation consists in the full characterization of weak binders. We have proved that weak binders still enjoy interesting semantic properties, comparably to what can be obtained through ν-binders. We have exploited weak binders to develop the static machinery (a type and effect system and a model checker) of a linguistic framework for resource usage control [1]. As a downside, we have found that weak binders, having a weaker structure than ν-binders, may make the life hard when going into the proofs.

The paper is organized as follows. We first introduce a calculus with explicit ν-binders, we give its operational and denotational semantics, and we show them fully abstract. We then remove ν-binders, and define a denotational semantics and an equational theory of weakly bound processes. Then, we define the bindify transformation, and we state its correctness: the bindification of a weakly bound process p is trace equivalent to p. After that, we compare the equational theories and the trace inclusion preorders of strongly bound and weakly bound processes. We conclude by reporting our experience about using weak binders, and by discussing some related work. Because of space limitations, here we shall omit the proofs of our statements. All the proofs are available in the companion technical report [2].

2 Strongly bound processes

We now introduce a process calculus with name binders, building upon Basic Process Algebras (BPAs, [3]). Our calculus shares with BPAs the primitives for sequential composition, for non-deterministic choice, and for recursion (though with a slightly different syntax). Quite differently from BPAs, our atomic actions (called *events*) have a parameter, which indicates the *resource* upon which the action is performed. Resources $r, r', \ldots \in \mathsf{Res}$ are system objects that can either be already available in the environment or be created at run-time. Resources can be accessed through a given finite set of *actions* $\alpha, \alpha', new, \ldots \in \mathsf{Act}$. The special action *new* represents the creation of a fresh resource: this means that for each dynamically created resource r, the event $new(r)$ must precede any other $\alpha(r)$. [1] An *event* $\alpha(r) \in \mathsf{Ev}$ abstracts from accessing the resource r through the action α. We also have events the target of which is a *name* $n, n', \ldots \in \mathsf{Nam}$, to be bound by an outer ν. Since the name binders are explicit in this calculus, we call its processes *strongly bound*, whose abstract syntax is given in Def. 2.1.

Definition 2.1 Syntax of strongly bound processes

P, Q	$::=$	ε	empty process
		h	variable
		$\alpha(\rho)$	event ($\rho \in \mathsf{Res} \cup \mathsf{Nam}$)
		$\nu n.P$	resource binding
		$P \cdot Q$	sequential composition
		$P + Q$	choice
		$\mu h.P$	recursion

In a recursion $\mu h.P$, the free occurrences of h in P are bound by μ. In the construct $\nu n.\, P$, the ν acts as a binder for the free occurrences of the name n in P. The intended meaning is to keep track of the binding between n and a freshly created resource. A process is *closed* when it has no free names and variables.

The behaviour of a strongly bound process is described by the set of sequential traces (typically denoted by $\eta, \eta', \ldots \in \mathsf{Ev}^*$) of its events. As usual, ε denotes the empty trace, and $\varepsilon\eta = \eta = \eta\varepsilon$. The trace semantics $[\![P]\!]^{op}$ of a closed, strongly bound process P, is a function from finite set of resources to sets of traces (Def. 2.2). We first introduce an auxiliary labelled transition relation $P, \mathcal{R} \xrightarrow{a} P', \mathcal{R}'$ (where $a \in \mathsf{Ev} \cup \{\varepsilon\}$ and $\mathcal{R}, \mathcal{R}' \subset \mathsf{Res}$). The set $\mathcal{R}$ in configurations accumulates the resources created at run-time, so that no resource can be created twice, e.g.

$$
\begin{aligned}
(\nu n.\, new(n)) \cdot (\nu n.\, new(n)), \emptyset &\xrightarrow{\varepsilon} new(r_0) \cdot (\nu n.\, new(n)), \{r_0\} \\
&\xrightarrow{new(r_0)} \nu n.\, new(n), \{r_0\} \\
&\not\longrightarrow new(r_0), \{r_0\}
\end{aligned}
$$

[1] We conjecture this is a decidable property, e.g. suitably adapting the techniques of [10] should enable us to identify and discard those P that produce ill-formed traces where an $\alpha(r)$ comes before a $new(r)$.

The labelled transition relation is then exploited in the definition of $[\![P]\!]^{op}$, which contains two kinds of traces. First, we include in $[\![P]\!]^{op}$ all the traces for *terminating* executions, i.e. those leading to ε. Then, we add all the prefixes of *all* executions, and mark these truncated traces with a trailing ! symbol. Here, we just let ! be a distinguished event not in Ev. Including these $\eta!$ prefixes in $[\![P]\!]^{op}$ is useful, since they allow us to observe non-terminating computations.

Definition 2.2 Trace semantics of strongly bound processes

$$\alpha(r),\mathcal{R} \xrightarrow{\alpha(r)} \varepsilon,\mathcal{R}\cup\{r\} \qquad \nu n.\,P,\mathcal{R} \xrightarrow{\varepsilon} P\{r/n\},\mathcal{R}\cup\{r\} \quad \text{if } r\notin\mathcal{R}$$

$$\varepsilon\cdot P,\mathcal{R} \xrightarrow{\varepsilon} P,\mathcal{R} \qquad P\cdot Q,\mathcal{R} \xrightarrow{a} P'\cdot Q,\mathcal{R}' \text{ if } P,\mathcal{R} \xrightarrow{a} P',\mathcal{R}'$$

$$P+Q,\mathcal{R} \xrightarrow{\varepsilon} P,\mathcal{R} \qquad P+Q,\mathcal{R} \xrightarrow{\varepsilon} Q,\mathcal{R} \qquad \mu h.\,P,\mathcal{R} \xrightarrow{\varepsilon} P\{\mu h.\,P/h\},\mathcal{R}$$

The trace semantics $[\![P]\!]^{op}(\mathcal{R})$ is then defined as

$$[\![P]\!]^{op}(\mathcal{R}) = \{\,\eta \mid P,\mathcal{R} \xrightarrow{\eta} \varepsilon,\mathcal{R}'\,\} \cup \{\,\eta! \mid P,\mathcal{R} \xrightarrow{\eta} P',\mathcal{R}'\,\}$$

Example 2.3 Consider the following strongly bound processes:

$$P_0 = \mu h.\,\alpha(r)\cdot h \qquad P_1 = \mu h.\,h\cdot\alpha(r) \qquad P_2 = \mu h.\,\nu n.\,(\varepsilon+\alpha(n)\cdot h)$$

Then, $[\![P_0]\!]^{op}(\emptyset) = \alpha(r)^*!$, i.e. P_0 generates traces with an arbitrary, finite number of $\alpha(r)$. Note that all the traces of P_0 are non-terminating (as indicated by the !) since there is no way to exit from the recursion. Instead, $[\![P_1]\!]^{op}(\emptyset) = \{!\}$, i.e. P_1 loops forever, without generating any events. The semantics of $[\![P_2]\!]^{op}(\emptyset)$ consists of all the traces of the form $\alpha(r_1)\cdots\alpha(r_k)$ or $\alpha(r_1)\cdots\alpha(r_k)!$, for all $k \geq 0$ and pairwise distinct resources r_i. □

The denotational semantics $[\![P]\!]^s_\theta$ of a strongly bound process P is given below (Def. 2.5) as a function Y in a cpo D_s, which we define now. We first let D_0 be $\{\,X \subseteq \mathsf{Ev}^* \cup \mathsf{Ev}^*! \mid ! \in X\,\}$, that is the cpo of sets X of traces such that $! \in X$. Then we let D_h be the cpo $\mathcal{P}_{fin}(\mathsf{Res}) \rightharpoonup D_0$ (where $\rightharpoonup$ denotes partiality). Finally, D_s is the cpo $(\mathsf{Nam} \to \mathsf{Res}) \to D_h$ Intuitively, $[\![P]\!]^s_\theta(\chi)(\mathcal{R})$ contains all the possible traces of P. The first argument $\chi \in \mathsf{Nam} \to \mathsf{Res}$ records the bindings between names and resources. The second argument $\mathcal{R} \in \mathcal{P}_{fin}(\mathsf{Res})$ is a finite set of resources which indicates those already used, so to make them unavailable for future creations. As usual, the parameter θ binds the free variables of P (in our case, to values in D_h).

Before giving the semantics, it is convenient to introduce some auxiliary definitions that help in composing traces sequentially (see Def. 2.4 below).

The operator $\odot$ ensures that all the events after a ! are discarded. For instance, the process $P = (\mu h.\,h)\cdot\alpha(r)$ will never fire the event $\alpha(r)$, because of the infinite loop that precedes the event. The composition of the semantics of the first component $\mu h.\,h$ is $\{!\}$, while the semantics of $\alpha(r)$ is $\{!, \alpha(r), \alpha(r)!\}$. Combining the two semantics results in $\{!\} \odot \{!, \alpha(r), \alpha(r)!\} = \{!\}$.

The operator $\boxdot$ takes two semantics and combines their traces sequentially. While doing that, it records the resources created, so to avoid that a resource is

generated twice. For instance, let $P = (\nu n.\, new(n)) \cdot (\nu n'.\, new(n'))$. The traces of the right-hand side $\nu n'.\, new(n')$ must not generate the same resources used in the left-hand side $\nu n.\, new(n)$, e.g. $new(r_0)new(r_0)$ is *not* a possible trace of P.

The definition of $\boxdot$ exploits the auxiliary operator R, that computes the set of resources occurring in a trace η. Also, $\downarrow \in \mathsf{R}(\eta)$ indicates that η is terminating, i.e. it does not contain any !s.

Definition 2.4 Let $X \in D_0$, and $x \in \mathsf{Ev} \cup \{!\}$. We define $x \odot X$ and $\eta \odot X$ as:

$$x \odot X = \begin{cases} \{\, x\,\eta \mid \eta \in X \,\} & \text{if } x \neq ! \\ \{x\} & \text{if } x = ! \end{cases} \qquad (a_1 \cdots a_n) \odot X = a_1 \odot \cdots \odot a_n \odot X$$

Given $Y_0, Y_1 \in D_s$, their composition $Y_0 \boxdot Y_1$ is:

$$Y_0 \boxdot Y_1 = \lambda \chi, \mathcal{R}.\ \bigcup \{\, \eta_0 \odot Y_1(\chi)(\mathcal{R} \cup \mathsf{R}(\eta_0)) \mid \eta_0 \in Y_0(\chi)(\mathcal{R}) \,\}$$

where $\mathsf{R}(\eta)$ is defined inductively as follows:

$$\mathsf{R}(\varepsilon) = \{\downarrow\} \qquad \mathsf{R}(\eta\,\alpha(r)) = \mathsf{R}(\eta) \cup \{r\} \ \text{ if } ! \notin \eta \qquad \mathsf{R}(\eta\,!\,\eta') = \mathsf{R}(\eta) \setminus \{\downarrow\}$$

Definition 2.5 Denotational semantics of strongly bound processes

$$[\![\varepsilon]\!]^s_\theta = \lambda \chi, \mathcal{R}.\, \{!, \varepsilon\} \qquad [\![h]\!]^s_\theta = \lambda \chi, \mathcal{R}.\, \theta(h)(\mathcal{R})$$

$$[\![\alpha(\rho)]\!]^s_\theta = \lambda \chi, \mathcal{R}. \begin{cases} \{!, \alpha(\rho), \alpha(\rho)\,!\} & \text{if } \rho = r \\ \{!, \alpha(\chi(n)), \alpha(\chi(n))\,!\} & \text{if } \rho = n \end{cases} \qquad [\![P \cdot Q]\!]^s_\theta = [\![P]\!]^s_\theta \boxdot [\![Q]\!]^s_\theta$$

$$[\![\nu n.\, P]\!]^s_\theta = \lambda \chi, \mathcal{R}.\ \textstyle\bigcup_{r \notin \mathcal{R}} [\![P]\!]^s_\theta(\chi\{r/n\})(\mathcal{R} \cup \{r\}) \qquad [\![P + Q]\!]^s_\theta = [\![P]\!]^s_\theta \sqcup [\![Q]\!]^s_\theta$$

$$[\![\mu h.P]\!]^s_\theta = \lambda \chi, \mathcal{R}.\ \textstyle\bigcup_{i \geq 0} \left(\lambda Z.\, \lambda \bar{\mathcal{R}}.\, [\![P]\!]^s_{\theta\{Z/h\}}(\chi)(\bar{\mathcal{R}})\right)^i (\lambda \mathcal{R}.\{!\})\ (\mathcal{R})$$

The semantics of an event $\alpha(r)$ comprises the possible "truncations" of $\{\alpha(r)\}$, i.e. $!, \alpha(r)\,!$ and $\alpha(r)$ (notice that ! is always included in the semantics of all P, coherently with the definition of the trace semantics). The semantics of $\alpha(n)$ is similar, but it looks in χ for the resource associated with n. The semantics of $\nu n.\, P$ joins the semantics of P, where the parameters $\mathcal{R}$ and χ are updated to record the binding of n with r, for *all* the resources r not yet used in $\mathcal{R}$. The semantics of $P \cdot Q$ combines the semantics of P and Q with the operator $\boxdot$. The semantics of $P + Q$ is the least upper bound of the semantics of P and Q. The semantics of a recursion $\mu h.\, P$ is the least upper bound of $f^i(\lambda \mathcal{R}.\{!\})$, where $f(Z) = \lambda \bar{\mathcal{R}}.\, [\![P]\!]^s_{\theta\{Z/h\}}(\chi)(\bar{\mathcal{R}})$. Since f is continuous and $\lambda \mathcal{R}.\{!\}$ is the bottom element of the cpo D_h, then $f^i(\lambda \mathcal{R}.\{!\})$ is an ω-chain, and its least upper bound is the least fixed point of f.

The following theorem states that the denotational semantics of strongly bound processes is fully abstract with respect to their operational semantics.

Theorem 2.6 (Full abstraction) *Let $\mathcal{R}$ be a finite sets of resources, and let $\emptyset$ be the empty mapping. Then, for all closed strongly bound processes P:*

$$[\![P]\!]^{op}(\mathcal{R}) \ = \ [\![P]\!]^s_\emptyset(\emptyset)(\mathcal{R})$$

3 Weakly bound processes

In strongly bound processes, the ν-binders precisely define the scope of names. However, classical equational theories [11] for these processes usually allow binders to be floated out, towards the top-level, e.g. in $P_0 + \nu n.\, P_1 = \nu n.\, P_0 + P_1$, under the usual conditions. Indeed, the binder can always be brought outside a context, provided that 1) no recursion boundary is crossed, i.e. in $\mu h.\, \nu n.P$ the binder cannot be moved outside, and 2) no name in the context is captured. Because of this, it is often convenient to define a *normal form* for processes, where all the binders are placed at their top-most position, i.e. at the top-level or just under a recursion. These are standard and well-known facts about process algebras.

One might wonder what information is actually carried by the presence of the ν-binders. From an operational point of view, we can see them as the points where resources are created. In our setting, this information is also carried by the *new* events. Therefore, it is interesting to study whether, under this assumption, we can neglect placing binders in our processes, and let the *new* events to define, at least in some loose way, the scope of names.

To this purpose, we now introduce *weakly bound* processes, which have no ν-binders (Def. 3.1). For instance, let $p = \mathit{new}(n) \cdot \alpha(n) + \mathit{new}(m) \cdot \alpha'(m)$. Here, the event $\mathit{new}(n)$ binds the name n, while $\mathit{new}(m)$ binds m. We shall later on define a semantics of weakly bound processes such that p is equivalent to the strongly bound process $(\nu n.\mathit{new}(n) \cdot \alpha(n)) + (\nu m.\mathit{new}(m) \cdot \alpha'(m))$, as the intuition suggests.

While weakly bound processes may make our reasoning more agile, we must not neglect that, unlike in the strongly bound case, weakly bound processes are possible where names have no clear scope. E.g., in $\mathit{new}(n) \cdot (\mathit{new}(n) + \varepsilon) \cdot \alpha(n)$ it is not clear what binds the last occurrence of n. Roughly, these troublesome processes are those that can be derived from a strongly bound process by neglecting to α-convert some name while enlarging the scope of a ν-binder, yielding to unwanted name captures. We shall return to this point in Sect. 4.

Definition 3.1 Syntax of weakly bound processes

p, q	$::=$	ε	empty process
		h	variable
		$\alpha(\rho)$	event ($\rho \in \mathsf{Res} \cup \mathsf{Nam}$)
		$\mathit{new}(n)$	resource creation
		$p \cdot q$	sequential composition
		$p + q$	choice
		$\mu h.p$	recursion

Free names in weakly bound processes have to be dealt with quite peculiarly, because of the absence of ν-binders. Consider e.g. $p = p' \cdot \alpha(n)$. To tell whether n is free in p we have to inspect p'. For example if $p' = \mathit{new}(n)$, we shall consider n as non-free; instead, if $p' = \varepsilon$, the name n is obviously free. Given p', we define which names are *bound* by p', so to extend the scope of the names of p' when it occurs at the left of another process, as in $p' \cdot p''$. Non-determinism complicates

matters: it might happen than a process p' binds a name to a resource only in some, but not all, of its execution, e.g. $p' = new(n) + \varepsilon$. So, we define two sets of names, the *must-bound* names $bn^{\Box}(p)$ and the *may-bound* names $bn^{\diamond}(p)$, for the names that are bound in *every* execution of p, and the names that are bound in *some* execution of p, respectively (see Def. 3.2). So, if $p' = new(n) + \varepsilon$, we have $bn^{\Box}(p') = \emptyset$ and $bn^{\diamond}(p') = \{n\}$. Note that the sets $bn^{\Box}(p')$ and $bn^{\diamond}(p')$ can be seen as static approximations for the actual run-time bindings created by the process p'. Of course, $bn^{\Box}(p) \subseteq bn^{\diamond}(p)$. Note that no "weak" binding can escape a recursion, as real ν-binders cannot cross recursive contexts. So, in $(\mu h.\, new(n) \cdot h + \varepsilon) \cdot \alpha(n)$ the last n is free, and is unrelated to the $new(n)$ event under the μh. Therefore, the bound names (both *must* and *may*) of a recursion are empty.

Definition 3.2 Must-bound names $\mathbf{bn^{\Box}(p)}$ and may-bound names $\mathbf{bn^{\diamond}(p)}$

$$bn^{\Box}(\varepsilon) = bn^{\Box}(h) = \emptyset \qquad bn^{\Box}(\alpha(\rho)) = \begin{cases} \{n\} & \text{if } \alpha = new \text{ and } \rho = n \\ \emptyset & \text{otherwise} \end{cases}$$

$$bn^{\Box}(p \cdot q) = bn^{\Box}(p) \cup bn^{\Box}(q) \qquad bn^{\Box}(p + q) = bn^{\Box}(p) \cap bn^{\Box}(q) \qquad bn^{\Box}(\mu h.\, p) = \emptyset$$

$$bn^{\diamond}(p) = \begin{cases} bn^{\Box}(p) & \text{if } p = \varepsilon,\ h,\ \alpha(\rho),\ \mu h.\, p' \\ bn^{\diamond}(p') \cup bn^{\diamond}(p'') & \text{if } p = p' + p'' \text{ or } p = p' \cdot p'' \end{cases}$$

We can now define the free names $fn(p)$ of a weakly bound process p. This is mostly standard, except that must-bound names are checked to single out captured names. The choice of using must-bound names instead of may-bound names is done so that, e.g. in $p = (new(n) + \varepsilon) \cdot \alpha(n)$ we consider n as free. This has the nice property that, whenever $fn(p) = \emptyset$, in no execution of p we will attempt to fire an event $\alpha(n)$ without a proper binding for n.

Definition 3.3 Free names $\mathbf{fn(p)}$

$$fn(h) = \emptyset \qquad fn(\alpha(\rho)) = \begin{cases} \{n\} & \text{if } \rho = n \text{ and } \alpha \neq new \\ \emptyset & \text{otherwise} \end{cases} \qquad fn(\mu h.\, p) = fn(p)$$

$$fn(\varepsilon) = \emptyset \qquad fn(p \cdot q) = fn(p) \cup (fn(q) \setminus bn^{\Box}(p)) \qquad fn(p + q) = fn(p) \cup fn(q)$$

We now define a denotational semantics of weakly bound processes. Unlike in the case of strongly bound processes, where the result of the semantics was a set of event traces, here we also need to keep track of the bindings generated by the *new* events. We shall then use sets of pairs (η, χ) instead of sets of traces η. Note that this difference – the extra χ – between the semantic domains for the strongly/weakly bound processes is exactly the same difference between the classic domains for programming languages with static/dynamic scoping.

As we did with strongly bound processes in Def. 2.4, we introduce the auxiliary operators $\odot$ and $\boxdot$ to handle sequential composition.

The operator $\odot$ merges two pairs (η, χ), so ensuring that all the events after a $!$ are discarded, as well as the bindings created after the $!$. For example, $(\eta!, \chi) \odot (\eta', \chi') = (\eta!, \chi)$, discarding both η' and χ'. Here we also use two cpos, D_1 and D_w, to play the role of D_0 and D_s used for strongly bound processes. We let D_1 be the cpo of sets X of pairs (η', χ') such that there exists a pair in X with $\eta' = !$. Formally, D_1 is the cpo $\{\, X \subseteq (\mathsf{Ev}^* \cup \mathsf{Ev}^*\,!) \times (\mathsf{Nam} \to \mathsf{Res}) \mid \exists \chi'.\, (!, \chi') \in X \,\}$.

Definition 3.4 Let $a \in \mathsf{Ev} \cup \{!\}$, $X \in D_1$, $(\eta, \chi), (\eta', \chi') \in X$. We define $\odot$ as:

$$(a, \chi) \odot (\eta', \chi') = \begin{cases} (a, \chi) & \text{if } a = ! \\ (a\eta', \chi') & \text{otherwise} \end{cases}$$

$$(\eta, \chi) \odot (\eta', \chi') = (a_1, \chi) \odot \cdots \odot (a_k, \chi) \odot (\eta', \chi') \;\text{ if } \eta = a_1 \cdots a_k$$

$$(\eta, \chi) \odot X = \{\, (\eta, \chi) \odot (\bar\eta, \bar\chi) \mid (\bar\eta, \bar\chi) \in X \,\}$$

The operator $\boxdot$ takes two semantics Y_0 and Y_1 and combines their traces sequentially. In $Y_0 \boxdot Y_1$ the bindings (i.e. the χ) generated by Y_0 are passed to Y_1, so that e.g. $\mathit{new}(n) \cdot \alpha(n)$ works as expected.

Definition 3.5 Let $D_w = (\mathsf{Nam} \to \mathsf{Res}) \to \mathcal{P}_{\mathit{fin}}(\mathsf{Res}) \rightharpoonup D_1$ be the cpo of functions from functions from names to resources, to the finite subsets of Res to D_1 (where $\rightharpoonup$ denotes partiality). Given $Y_0, Y_1 \in D_w$, their composition $Y_0 \boxdot Y_1$ is:

$$Y_0 \boxdot Y_1 = \lambda \chi, \mathcal{R}.\ \bigcup \{\, (\eta_0, \chi_0) \odot Y_1(\chi_0)(\mathcal{R} \cup \mathsf{R}(\eta_0)) \mid (\eta_0, \chi_0) \in Y_0(\chi)(\mathcal{R}) \,\}$$

The denotational semantics $[\![p]\!]^w_\theta$ of a weakly bound process p is defined as a function $Y \in D_w$, where we assume that $Y(\chi)(\mathcal{R})$ is defined only if $\mathcal{R} \supseteq \mathit{ran}(\chi)$. The parameter θ is a mapping from the free variables h of p to D_h.

Definition 3.6 Denotational semantics of weakly bound processes

Below, we let $\mathit{set}_\chi I = \{\, (\eta, \chi) \mid \eta \in I \,\}$.

$$[\![\varepsilon]\!]^w_\theta = \lambda \chi, \mathcal{R}.\, \mathit{set}_\chi\{!, \varepsilon\} \qquad [\![h]\!]^w_\theta = \lambda \chi, \mathcal{R}.\, \mathit{set}_\chi \theta(h)(\mathcal{R})$$

$$[\![\alpha(\rho)]\!]^w_\theta = \lambda \chi, \mathcal{R}. \begin{cases} \mathit{set}_\chi\{!, \alpha(\rho), \alpha(\rho)\,!\} & \text{if } \rho = r \\ \mathit{set}_\chi\{!, \alpha(\chi(n)), \alpha(\chi(n))\,!\} & \text{if } \rho = n \in \mathit{dom}(\chi) \\ \{(!, \chi)\} \cup \bigcup_{r \notin \mathcal{R}} \mathit{set}_{\chi\{r/n\}}\{\alpha(r), \alpha(r)\,!\} & \text{if } \rho = n \notin \mathit{dom}(\chi) \text{ and } \alpha = \mathit{new} \end{cases}$$

$$[\![p \cdot q]\!]^w_\theta = [\![p]\!]^w_\theta \boxdot [\![q]\!]^w_\theta \qquad [\![p + q]\!]^w_\theta = [\![p]\!]^w_\theta \sqcup [\![q]\!]^w_\theta$$

$$[\![\mu h.p]\!]^w_\theta = \lambda \chi, \mathcal{R}.\, \mathit{set}_\chi \bigcup_{i \geq 0} \big(\lambda Z. \lambda \bar{\mathcal{R}}.\, \mathit{fst}([\![p]\!]^w_{\theta\{Z/h\}}(\chi|_{\mathit{dom}(\chi) \setminus bn^\diamond(p)})(\bar{\mathcal{R}}))\big)^i (\lambda \mathcal{R}.\{!\})\ (\mathcal{R})$$

The semantics above is similar to the one for strongly bound processes, so we just comment on the differences. First, each trace η has now been bundled with its generated bindings χ. Related to this, now the $\mathit{new}(n)$ event creates the actual binding, which augments the χ at hand. Note that we assume the operators $\cup$ and $\sqcup$ to be undefined when one of the arguments is undefined. This must hold also

for $\odot$ and $\boxdot$, so making e.g. the semantics of $(new(n) + \varepsilon) \cdot \alpha(n)$ undefined when $n \notin dom(\chi)$, since in one branch $\alpha(n)$ is evaluated without a proper binding for n.

The semantics of recursion variables h is peculiar. First, note that we chose the semantics parameter θ so that $\theta(h)$ is an element of D_h and not of D_w. This is because, when recursion is involved, the bindings of names must not be propagated: this is strictly related to the fact that ν-binders cannot cross a recursive context in strongly bound processes. For example, in the strongly bound process $\mu h.\, \nu n.\, P \cdot h \cdot P'$ there is no way for the resource bound to n to be "passed" to the inner "call" to h; similarly, if the inner "call" generates a binding, it cannot be "returned" so to interfere with P'. Of course, this would change if we allowed a more complex form of recursion where h can take a resource as an argument. Returning to the semantics of h, since $\theta(h) \in D_h$ needs no χ, then it suffices to pass it an $\mathcal{R}$, and then augment the returned set of traces with χ. This is accomplished by the set_χ function.

The semantics of the recursion $\mu h.\, p$ is quite similar to the one for strongly bound processes. For the reasons explained above, we compute a fixed point over D_h and not D_w. This means that we have to adapt the semantics of p, which is in D_w, to a function in D_h. More concretely, we just need to provide χ to $[\![p]\!]^w$ and ignore the χ returned by it. The latter is done by a trivial left projection, the *fst* in the actual formula. The χ we pass, instead, is the top-level χ – the one provided to the whole recursive process – after the bindings which affect $bn^\diamond(p)$ have been filtered out. This filtering is needed to prevent name confusion e.g. in $new(n) \cdot (\mu h.\, new(n) \cdot p)$, where the outer n is unrelated to the inner one. Aside from this, the fixed point is computed exactly as for the strongly bound processes, exploiting the continuity of $f(Z) = \lambda\bar{\mathcal{R}}.\, fst([\![p]\!]^w_{\theta\{Z/h\}}(\chi')(\bar{\mathcal{R}}))$.

4 Bindifying weakly bound processes

To make precise the scope of names in weakly bound processes, we shall translate them into strongly bound processes, through the transformation *bindify* (Def. 4.3). This transformation will insert the ν-binders at the right points, provided that the introduced scopes of names do not interfere dangerously. We shall call *well-bound* those weakly bound processes that can be safely translated into strongly bound ones. To help the intuition, we shall first give some examples.

Example 4.1 Consider the weakly bound processes:

$$p_1 = new(n) \cdot new(n) \cdot \alpha(n) \qquad p_2 = \alpha(n) \cdot new(n) \qquad p_3 = new(n) + \alpha(n)$$

$$p_4 = (\varepsilon + new(n)) \cdot \alpha(n) \quad p_5 = \mu h.new(n).h \quad p_6 = new(n) \cdot (\mu h.\, (\varepsilon + new(n) \cdot h)) \cdot \alpha(n)$$

The processes p_1, p_2, p_3, p_4 are not well-bound. If p_1 were such, its bindification would either be $\nu n.\, new(n) \cdot (\nu n.\, new(n)) \cdot \alpha(n)$ – where α is performed on the resource generated by the outer ν-binder – or $\nu n.\, new(n) \cdot (\nu n.\, new(n) \cdot \alpha(n))$ – where α acts on the resource of the inner binder. Because of this possible ambiguity, we treat p_1 as not well-bound. The process p_2 is not well-bound, too, because it would produce an ill-formed trace $\alpha(r) new(r)$ where the event $\alpha(r)$ is fired *before* the event $new(r)$ that signals the creation of r. Similarly, the process p_3 is not well-bound, because its bindification would give rise to the ill-formed trace $\alpha(r)$. The process p_4 is not well-bound, because choosing the branch ε would lead to a similar situation. Observe

that the denotation of p_1 contains the non-sense trace $new(r)new(r)\alpha(r)$, while the semantics of p_2, p_3 and p_4 are undefined, because $\boxdot$ and $\sqcup$ are strict. Finally, the process p_5 is well-bound: it loops over $new(n)$, generating a fresh resource at each iteration. Also, p_6 is well-bound, because the μ-binder clearly separates the scope of the outer $new(n)$ from that of the inner one. □

The following definition formalizes when a process is well-bound. The empty process, variables and events are well-bound. A recursion is well-bound when its body is such. A choice $p + q$ is well-bound when both p and q are well-bound. Additionally, we require that the may-bound names of p are disjoint from the free names of q, and *viceversa* (e.g. $new(n) + \alpha(n)$ is not well-bound). A sequence $p \cdot q$ is well-bound when both p and q are well-bound, and furthermore (i) the may-bound names of q are disjoint from the names of p (e.g. $\alpha(n) \cdot new(n)$ and $new(n) \cdot new(n)$ are not well-bound), and (ii) the free names of q are either must-bound in p, or they are not may-bound in p (e.g. $(\varepsilon + new(n)) \cdot \alpha(n)$ is not well-bound).

Definition 4.2 Well-bound processes

A weakly bound process p is *well-bound* when $wb(p)$, defined inductively as:

$$wb(\varepsilon) = wb(h) = wb(\alpha(\rho)) = true \qquad wb(\mu h.\, p) \text{ if } wb(p)$$

$$wb(p + q) \text{ if } wb(p), wb(q), bn^{\diamond}(p) \cap fn(q) = bn^{\diamond}(q) \cap fn(p) = \emptyset$$

$$wb(p \cdot q) \text{ if } wb(p), wb(q), bn^{\diamond}(q) \cap (bn^{\diamond}(p) \cup fn(p)) = (bn^{\diamond}(p) \setminus bn^{\square}(p)) \cap fn(q) = \emptyset$$

We now introduce the *bindify* transformation, which is defined on well-bound processes only. The may-bound names are lifted to the leftmost position of the bindified process, and they are placed within the scope of a ν-binder. In the case of a recursion $\mu h.\, p$, the may-bound names of p are lifted to the leftmost position *within* the recursion, i.e. they do not escape the scope of the μh.

Definition 4.3 Bindification

If $wb(p)$, the bindification $bindify(p)$ of p is a strongly bound process, defined as:

$$bindify(p) = \nu\, bn^{\diamond}(p).\, \beta(p)$$

where the auxiliary operator β is defined inductively as follows:

$$\beta(\varepsilon) = \varepsilon \qquad \beta(\alpha(\rho)) = \alpha(\rho) \qquad \beta(p + q) = \beta(p) + \beta(q)$$

$$\beta(h) = h \qquad \beta(\mu h.\, p) = \mu h.\, bindify(p) \qquad \beta(p \cdot q) = \beta(p) \cdot \beta(q)$$

Example 4.4 Recall from Sect. 1 the process $p = new(n) \cdot \alpha(n) + new(m) \cdot \alpha(m)$. It is easy to check that p is well-bound, and that its may-bound names are:

$$bn^{\diamond}(p) = bn^{\diamond}(new(n) \cdot \alpha(n)) \cup bn^{\diamond}(new(m) \cdot \alpha(m)) = \{n, m\}$$

Then the bindification of p is the strongly bound process:

$$bindify(p) \;=\; \nu n.\nu m.(new(n) \cdot \alpha(n) + new(m) \cdot \alpha(m))$$

Example 4.5 Recall the process $p_5 = new(n) \cdot (\mu h.\,(\varepsilon + new(n) \cdot h)) \cdot \alpha(n)$ from Ex. 4.1. It is easy to check that p_5 is well-bound. Its may-bound names are:

$$bn^\diamond(p_5) = bn^\diamond(new(n)) \cup bn^\diamond(\mu h.\,(\varepsilon + new(n) \cdot h)) \cup bn^\diamond(\alpha(n)) = \{n\} \cup \emptyset = \{n\}$$

The bindification of p_5 is then computed as follows:

$$\begin{aligned} bindify(p_5) &= \nu n.\,\beta\big(new(n) \cdot (\mu h.\,(\varepsilon + new(n) \cdot h)) \cdot \alpha(n)\big) \\ &= \nu n.\,\big(\beta(new(n)) \cdot \mu h.\,bindify(\varepsilon + new(n) \cdot h) \cdot \beta(\alpha(n))\big) \\ &= \nu n.\,new(n) \cdot (\mu h.\,\nu n.\,\beta(\varepsilon + new(n) \cdot h)) \cdot \alpha(n) \\ &= \nu n.\,new(n) \cdot (\mu h.\,\nu n.\,(\varepsilon + new(n) \cdot h)) \cdot \alpha(n) \end{aligned}$$

We now state the correctness of bindification (Theorem 4.6). The "strong" semantics of $bindify(p)$ contains exactly the traces of the "weak" semantics of p.

Theorem 4.6 *For all closed, weakly bound processes p such that $wb(p)$, $[\![p]\!]^w_\emptyset(\emptyset)(\emptyset)$ is defined, and:*

$$[\![bindify(p)]\!]^s_\emptyset(\emptyset)(\emptyset) \quad = \quad fst([\![p]\!]^w_\emptyset(\emptyset)(\emptyset))$$

5 Equational theories and trace inclusion

In this section we provide strongly bound and weakly bound processes with an equational theory and a trace inclusion preorder. We shall state their correctness, i.e. the equational theory preserves the set of traces, while the preorder preserves their inclusion. Finally, we shall highlight some differences between the two calculi.

We first give in Def. 5.1 an equational theory of strongly bound processes.

Definition 5.1 An equational theory of strongly bound processes

The relation $=$ over strongly bound processes is the least congruence including α-conversion of names and variables such that:

$$P + P = P \qquad (P + P') + P'' = P + (P' + P'') \qquad P + P' = P' + P$$

$$(P \cdot P') \cdot P'' = P \cdot (P' \cdot P'') \qquad \varepsilon \cdot P = P = P \cdot \varepsilon$$

$$(P + P') \cdot P'' = P \cdot P'' + P' \cdot P'' \qquad P \cdot (P' + P'') = P \cdot P' + P \cdot P''$$

$$\mu h.\mu h'.P = \mu h'.\mu h.P \qquad \mu h.P = P\{\mu h.\,P/h\} \qquad \nu n.\varepsilon = \varepsilon$$

$$\nu n.\nu n'.P = \nu n'.\nu n.P \qquad \nu n.(P + P') = (\nu n.P) + P' \ \text{ if } n \notin fn(P')$$

$$\nu n.(P \cdot P') = P \cdot (\nu n.P') \ \text{ if } n \notin fn(P) \qquad \nu n.(P \cdot P') = (\nu n.P) \cdot P' \ \text{ if } n \notin fn(P')$$

The operation $+$ is associative, commutative and idempotent; $\cdot$ is associative, has identity ε, and distributes over $+$. The binders μ and ν allow for α-conversion of bound names and variables, and can be rearranged. A μh can be introduced/eliminated when h does not occur free. A νn can be extruded when it does not bind a free occurrence of n. A $\mu h.\,P$ can be folded/unfolded as usual.

As expected, the equational theory above is not complete, e.g. $[\![P]\!]^s = [\![P']\!]^s$ does not imply $P = P'$. E.g., $\mu h.\, \alpha(r) \cdot h$ cannot be equated to $\mu h.\, \alpha(r) \cdot \alpha(r) \cdot h$, yet they have the same traces $\alpha(r)^*\,!$. However, the equational theory is sound w.r.t. our semantics, as established by the first item Theorem 5.3 below.

We then define a preorder $P \preceq Q$ betweeen strongly bound processes. The preorder $\preceq$ includes equivalence, and it is closed under contexts. Also, a strongly bound process P can be arbitrarily "weakened" to $P + Q$.

Definition 5.2 A trace inclusion preorder of strongly bound processes

The relation $\preceq$ over strongly bound processes is the least precongruence such that:

$$P \preceq Q \quad \text{if } P = Q \qquad\qquad P \preceq P + Q$$

The following theorem states that the equational theory $=$ and the preorder $\preceq$ agree with the semantics of strongly bound processes.

Theorem 5.3 *For all closed, strongly bound processes P and Q:*

- *if $P = Q$, then $[\![P]\!]^s_\emptyset = [\![Q]\!]^s_\emptyset$.*
- *if $P \preceq Q$ then $[\![P]\!]^s_\emptyset(\chi)(\mathcal{R}) \subseteq [\![Q]\!]^s_\emptyset(\chi)(\mathcal{R})$, for all $\mathcal{R}$ and χ.*

We now consider how to express an equational theory and a trace inclusion preorder for weak binders, in the same spirit of Def. 5.1 and Def. 5.2. In spite of their weaker structure, weakly bound processes still share many semantic-preserving equational properties with strongly bound processes, as summarized in Def. 5.4. Notably, the equations involving $+$ and $\cdot$ are identical with respect to Def. 5.1. The recursions μh can be rearranged, as before. Of course, here we do not have ν-binders, so the α-conversion of bound names can not be done, in general. As an important exception, we know that bound names inside a recursion can not escape, so their scope is completely known. In this case, we allow for α-conversion. Note that unfolding recursions is not allowed, otherwise we would have $\mu h.new(n) \cdot h \approx new(n) \cdot (\mu h.new(n) \cdot h) \approx new(n) \cdot new(n) \cdot (\mu h.new(n) \cdot h)$, so causing name confusion — indeed, the first two processes are well-bound, while the last one is not. As with strong binders, the equational theory below is not complete, yet it is sound w.r.t. the $[\![-]\!]^w$ semantics, as established by the first item of Theorem 5.7.

Definition 5.4 An equational theory of weakly bound processes

The relation $\approx$ over weakly bound processes is the least congruence including α-conversion of variables such that:

$$p + p \approx p \qquad (p + p') + p'' \approx p + (p' + p'') \qquad p + p' \approx p' + p \qquad \varepsilon \cdot p \approx p \approx p \cdot \varepsilon$$

$$(p \cdot p') \cdot p'' \approx p \cdot (p' \cdot p'') \quad (p + p') \cdot p'' \approx p \cdot p'' + p' \cdot p'' \quad p \cdot (p' + p'') \approx p \cdot p' + p \cdot p''$$

$$\mu h.\mu h'.p \approx \mu h'.\mu h.p \qquad \mu h.p \approx \mu h.(p\{m/n\}) \ \text{ if } n \in bn^\diamond(p) \text{ and } m \notin p$$

Example 5.5 The equational theories shown above offer an opportunity to compare strong ν-binders with weak *new* binders. Consider the following equation: $new(n) \cdot p + new(n) \cdot q \approx new(n) \cdot (p + q)$. This is a trivial fact, since it directly follows from the distributive law. Its equivalent for strongly bound processes, $(\nu n.\, P) + (\nu n.\, Q) = \nu n.\, (P + Q)$, appears instead to be non trivial. Indeed, although Def. 5.1 comprises all the classic equations for ν-binders, the mentioned equation can not be derived from them, since we can not identify the two binders. Yet, in most process algebras, we expect the equation to be sound w.r.t. any reasonable process equivalence relation. So, in this case weak binders offer a simpler view.

We shall now introduce a preorder $p \precsim_{\mathcal{N}} q$ on weakly bound processes. Here, we use a set of names $\mathcal{N}$ as an index to the preorder relation. This index is needed to avoid name confusion, as we shall see below. When $p \precsim_{\mathcal{N}} q$ holds, then the semantics of p is included in that of q (second item of Theorem. 5.7).

Definition 5.6 A trace inclusion preorder of weakly bound processes

The relation $\precsim_{\mathcal{N}}$ over weakly bound processes is the least preorder such that:

$$p \precsim_{\emptyset} q \quad \text{if } p \approx q \qquad p \precsim_{\emptyset} p + q \qquad p \precsim_{\mathcal{N} \cup \mathcal{N}'} p'' \quad \text{if } p \precsim_{\mathcal{N}} p' \text{ and } p' \precsim_{\mathcal{N}'} p''$$

$$\mathcal{C}(p) \precsim_{\mathcal{N}} \mathcal{C}(q) \quad \text{if } p \precsim_{\mathcal{N}} q \text{ and } \mathcal{N} \cap (bn^{\diamond}(\mathcal{C}) \cup fn(\mathcal{C})) = \emptyset$$

$$p\sigma\{\mu h.\, p/h\} \precsim_{ran(\sigma)} \mu h.\, p \quad \text{if } ran(\sigma) \cap fn(p) = \emptyset$$

where $\mathcal{C} = p \cdot \bullet \mid \bullet \cdot p \mid p + \bullet \mid \bullet + p$ is a *context*, $\sigma : \mathsf{Nam} \to \mathsf{Nam}$ is an injective function with $dom(\sigma) = bn^{\diamond}(p)$, and $p\sigma\{\mu h.\, p/h\}$ is capture-avoiding.

The preorder $\precsim_{\mathcal{N}}$ includes $\approx$-equivalence (Def. 5.4). A process p can be arbitrarily "weakened" to $p + q$. The relation is closed under contexts, provided that the names in $\mathcal{N}$ are disjoint from those in the context. Note that, because of this side condition, $\precsim_{\mathcal{N}}$ is not a precongruence, unlike $\preceq$ for strongly bound processes. Folding/unfolding is possible, but in a weaker form than in Def. 5.1. To avoid name confusion and preserve well-boundness, the unfolded names must be fresh. For instance, if $p = \mu h.\, new(n) \cdot \alpha(n) \cdot h$, then we shall have $new(n') \cdot \alpha(n') \cdot p \precsim_{\{n'\}} p$. The name n' in $\precsim_{\{n'\}}$ is needed to avoid name clashes. For instance, it prevents from using the previous unfolding in the context $\mathcal{C} = \bullet \cdot \alpha'(n')$, since the extruded $new(n')$ would bind the name n' in $\alpha'(n')$, as checked by the context rule above. The side condition on the rule for folding/unfolding is needed to ensure that all the processes smaller (w.r.t. $\precsim$) than a well-bound process are well-bound (Theorem 5.8). Omitting the disjointness condition between $fn(p)$ and the range of the substitution σ would lead to situations like $\alpha(n') \cdot new(n') \precsim_{\{n'\}} \mu h.\, \alpha(n') \cdot new(n)$, where the right-hand side is well-bound, while the left-hand side is not. Substitutions of names must be coherent with bindification, i.e. they must not affect names that would be put under a ν-binder by $\beta(-)$, e.g. $(new(n) \cdot \mu h.\, new(n))\{m/n\} = new(m) \cdot \mu h.\, new(n)$. Similarly, substitutions can trigger α-conversions to avoid name captures, e.g. $(\mu h.\, new(m) \cdot \alpha(n))\{m/n\} = \mu h.\, new(m') \cdot \alpha(m)$.

We now formally state that our syntactic preorder agrees with the semantics

of weakly bound processes, as it yields trace inclusion. Note that trace inclusion requires the two semantics to be defined. Otherwise we have $new(n) \cdot \mu h.\, \alpha(n) \precsim_\emptyset new(n) \cdot \mu h.\, (new(n) + \varepsilon) \cdot \alpha(n)$: when the branch ε is chosen in the right-hand side, we find $\chi'(n) = \chi|_{dom(\chi)\setminus\{n\}}(n)$, so $\alpha(n)$ cannot be evaluated, and the whole semantics is undefined (while the semantics of the left-hand side is always defined). Note however that is q is well-bound, then also p is such (Theorem 5.8), and so by Theorem 4.6 both the semantics are defined.

Theorem 5.7 *For all closed, weakly bound processes p and q:*

- *if $p \approx q$, then $[\![p]\!]^w_\emptyset = [\![q]\!]^w_\emptyset$.*
- *if $p \precsim_\mathcal{N} q$ and, then $fst([\![p]\!]^w_\emptyset(\chi)(\mathcal{R})) \subseteq fst([\![q]\!]^w_\emptyset(\chi)(\mathcal{R}))$, for all $\mathcal{R}$ and χ such that $dom(\chi) \cap \mathcal{N} = \emptyset$ and both the semantics are defined.*

The projection *fst* in the statement above is necessary. Consider e.g. $p = new(n) \precsim_{\{n\}} \mu h.\, new(m) = q$. Here, the semantics of p and q agree on the η components, i.e. the truncations of $new(r)$ with $r \notin \mathcal{R}$, but p will augment χ with the new binding $\{r/n\}$, unlike q which does not affect χ.

The next theorem guarantees that *bindify* is well-defined, i.e. it maps $\approx$-equivalent weakly bound processes to $=$-equivalent strongly bound processes. Moreover, processes smaller (w.r.t. $\precsim_\mathcal{N}$) than well-bound processes are well-bound.

Theorem 5.8 *For all weakly bound processes p and q:*

- *if $p \approx q$, then $wb(p)$ if and only if $wb(q)$.*
- *if $p \approx q$ and $wb(p)$, then $bindify(p) = bindify(q)$.*
- *if $p \precsim_\mathcal{N} q$ and $wb(q)$, then $wb(p)$.*

6 Conclusions

We have investigated weak binders – a construct for fresh name generation – as an alternative for ν-binders in nominal calculi. Weak binders allow for a looser reasoning, while still admitting a trace-preserving translation into strong binders. However, this comes at a cost: often, useful properties, e.g. trace inclusion (Th. 5.7), require more side conditions to be checked for ensuring sanity. Also, α-conversion of names can only be applied inside μ-binders. This is possible through the last rule of the equational theory in Def. 5.4. An alternative would be to always consider weakly bound processes modulo α-conversion within the μ-binders, at the cost of making some proofs (e.g. those that do not depend on $\approx$) more complex. A further downside of weak binders is that compositionality is reduced, since e.g. $wb(p)$ and $wb(q)$ do not automatically imply $wb(p \cdot q)$ which – if needed – must be established by exploiting further assumptions on the names of p and q. Future work would address the use of weak binders in other process calculi. Indeed, we expect that weak binders enjoy stronger properties in calculi without sequential composition (e.g. CCS [11]). Moreover, studying some relaxed variants of well-boundness could improve the applicability of weak binders.

In our experiments with weak binders, we also found they sometimes lead to intricate proofs, since particular care must be exercised with corner cases. For instance, handling recursion in an operational semantics for weakly bound processes

seems to be quite complex. Indeed, naïve unfolding causes name confusion, so one has to resort to either renaming all bound names so that they are indeed globally fresh, or to record the "call frames" (entering/leaving the body of a recursion) in a stack. Since we need to keep track of this, run-time configurations become more complex, and we found our operational semantics (not presented in this paper) to be too inconvenient to be used in proofs. Even when using the denotational semantics (Def. 3.6), we felt that writing inductive statements for weak binders required more trial-and-error steps, w.r.t. strong binders. However, in some occasions weak binders may become a more agile tool. For instance, they can be exploited to implement a type and effect inference algorithm for a calculus with side effects and explicit name binders (like [1]), on top of an existing algorithm for a calculus without binders. Each time a ν-binder is encountered, a fresh name is generated, similarly to fresh type variables in Hindley-Milner type inference. After solving the obtained type and effect constraints through unification, the resulting effect is bindified. Of course, this is not always possible, e.g. when the effect is not well-bound. Possible counter-measures consist in suitably extending let-polymorphism to ν-binders.

Related work. A number of formal techniques have been developed to handle binding and freshness of names. The permutation model of sets introduced by Fraenkel-Mostowski has led to an elegant and powerful mathematical theory of naming [8]. The key observation of this approach is that α-conversion, binding and freshness can be defined through name permutations (or swappings). For instance, the freshness axiom for a name of a computational entity (i.e. an object, a process, a context, *etc.*) is expressed by saying that the fresh name does not belong to the support of the computational entity. Notably, in the permutation model the support of computational entities is *finite*. This mathematical theory has been used to model early and late semantics of the π-calculus [9]. Also, it has driven the design of a functional language, FreshML [20], which includes primitive mechanisms for handling fresh bindable names. In FreshML freshness is managed by a *gensym()* primitive to dynamically generate names, and a primitive for permuting names. Our notion of weakly bound processes exploits the *gensym()* primitive without resorting to α-conversion. Indeed, the bindify trasformation singles out the names in the finite support of a weakly bound process. A monadic denotational semantics for FreshML has been used to handle freshness through a continuation monad on FM-sets [19]. This semantics allows for translating the usual domain-theoretic results in the context of FM-sets, and to use them to prove freshness-related properties. There is also a cost associated to α-converting names [7,15] which could be reduced e.g. by compiling strong binders into weak binders.

The $\lambda\nu$-calculus presented in [16] extends the pure λ-calculus with names. In contrast to λ-bound variables, nothing can be substituted for a name, yet names can be tested for equality. Reduction in $\lambda\nu$ is confluent, and it allows for deterministic evaluation. Furthermore, all the observational equivalences that hold in the pure λ-calculus still hold in $\lambda\nu$. This has the practical consequence that all the equational techniques for transforming and verifying pure functional programs are also applicable to programs with local names. Nominal techniques have been implicitly used for reasoning about the semantics of functional languages with local state in [17], to prove when two functional programs are equivalent in every evaluation context.

Binding and freshness of names have been a main concern in process calculi. History-Dependent automata [13,14] provide an automata-based model where states are equipped with name permutations to manage freshness and garbage collections of names. They automatically manage the creation and deallocation of names, while allowing for a compact representation of the system behaviour, by collapsing the states that only differ for the renaming of local names. The π-calculus is extended in [4] with an operational model where names are localized to their owners; each sequential process has its logical space on names and a local manager generates fresh names whenever necessary.

References

[1] Bartoletti, M., P. Degano, G. L. Ferrari and R. Zunino, *Types and effects for resource usage analysis*, in: *Foundations of Software Science and Computation Structures (FOSSACS)*, 2007.

[2] Bartoletti, M., P. Degano, G. L. Ferrari and R. Zunino, *Hard life with weak binders*, Technical Report TR-08-13, Dip. Informatica, Univ. Pisa (2008).

[3] Bergstra, J. A. and J. W. Klop, *Algebra of communicating processes with abstraction*, Theoretical Computer Science **37** (1985), pp. 77–121.

[4] Bodei, C., P. Degano and C. Priami, *Names of the π calculus agents handled locally*, Theoretical Computer Science **253** (2001), pp. 155–184.

[5] Bravetti, M. and G. Zavattaro, *Towards a unifying theory for choreography conformance and contract compliance*, in: *Software Composition*, 2007.

[6] Castagna, G., N. Gesbert and L. Padovani, *A theory of contracts for web services*, in: *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 2008.

[7] Fernández, M., I. Mackie and F.-R. Sinot, *Closed reduction: explicit substitutions without α-conversion*, Math. Structures Comput. Sci. **15** (2005), pp. 343–381.

[8] Gabbay, M. and A. M. Pitts, *A new approach to abstract syntax with variable binding*, Formal Asp. Comput. **13** (2002), pp. 341–363.

[9] Gabbay, M. J., *The pi-calculus in FM*, in: F. Kamareddine, editor, *Thirty-five years of Automath*, Applied Logic Series **28**, Kluwer, 2003 pp. 247–269.

[10] Igarashi, A. and N. Kobayashi, *Type reconstruction for linear -calculus with i/o subtyping*, Inf. Comput. **161** (2000), pp. 1–44.

[11] Milner, R., "Communication and concurrency," Prentice-Hall, Inc., 1989.

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

[13] Montanari, U. and M. Pistore, *An introduction to history dependent automata*, Electr. Notes Theor. Comput. Sci. **10** (1997).

[14] Montanari, U. and M. Pistore, *Structured coalgebras and minimal hd-automata for the π-calculus*, Theor. Comput. Sci. **340** (2005), pp. 539–576.

[15] Norrish, M. and R. Vestergaard, *Proof pearl: De bruijn terms really do work*, in: *TPHOLs*, 2007, pp. 207–222.

[16] Odersky, M., *A functional theory of local names*, in: *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 1994.

[17] Pitts, A. M. and I. D. B. Stark, *Operational reasoning for functions with local state*, Higher order operational techniques in semantics (1998), pp. 227–274.

[18] Sangiorgi, D. and D. Walker, "The π-Calculus: a Theory of Mobile Processes," Cambridge University Press, 2002.

[19] Shinwell, M. R. and A. M. Pitts, *On a monadic semantics for freshness*, Theoretical Computer Science **342** (2005), pp. 28–55.

[20] Shinwell, M. R., A. M. Pitts and M. Gabbay, *FreshML: programming with binders made simple*, in: *International Conference on Functional Programming (ICFP)*, 2003.