# SOS 2006

Preliminary Proceedings of the 3rd Workshop on

# Structural Operational Semantics

Bonn, Germany, 26th August 2006

Editors:
Rob van Glabbeek and Peter D. Mosses

# Contents

Preface v

Bartek Klin *(Invited Speaker)*
Bialgebraic methods in structural operational semantics ..... 1

MohammadReza Mousavi, Michel A. Reniers
On well-foundedness and expressiveness of promoted tyft ..... 13

Christiano Braga, Alberto Verdejo
Modular SOS with strategies ..... 25

Adrian Pop, Peter Fritzson *(Tool Demonstration)*
An Eclipse-based integrated environment for developing executable structural operational semantics specifications ..... 41

Robin Milner *(Joint Express-Infinity-SOS Invited Speaker)*
Local bigraphs and confluence: two conjectures ..... 46

Henrik Pilegaard, Flemming Nielson, Hanne Riis Nielson
Active evaluation contexts for reaction semantics ..... 55

Simone Tini
Notes on generative probabilistic bisimulation ..... 70

Vincent Danos, Jean Krivine, Fabien Tarissan
Self-assembling trees ..... 84

# Author Index

Braga, Christiano, 25

Danos, Vincent, 84

Fritzson, Peter, 41

Klin, Bartek, 1
Krivine, Jean, 84

Milner, Robin, 46
Mousavi, MohammadReza, 13

Nielson, Flemming, 55
Nielson, Hanne Riis, 55

Pilegaard, Henrik, 55
Pop, Adrian, 41

Reniers, Michel A., 13

Tarissan, Fabien, 84
Tini, Simone, 70

Verdejo, Alberto, 25

# Preface

This volume contains the preliminary proceedings of SOS 2006, the *Third Workshop on Structural Operational Semantics*, held on 26th August 2006 in Bonn, Germany, as a satellite event of CONCUR 2006, the *17th International Conference on Concurrency Theory*.

Structural operational semantics (SOS) provides a framework for giving operational semantics to programming and specification languages. A growing number of programming languages from commercial and academic spheres have been given usable semantic descriptions by means of structural operational semantics. Because of its intuitive appeal and flexibility, structural operational semantics has found considerable application in the study of the semantics of concurrent processes. Moreover, it is becoming a viable alternative to denotational semantics in the static analysis of programs, and in proving compiler correctness.

Recently, structural operational semantics has been successfully applied as a formal tool to establish results that hold for classes of process description languages. This has allowed for the generalization of well-known results in the field of process algebra, and for the development of a meta-theory for process calculi based on the realization that many of the results in this field only depend upon general semantic properties of language constructs.

This workshop aims at being a forum for researchers, students and practitioners interested in new developments, and directions for future investigation, in the field of structural operational semantics. One of the specific goals of the workshop is to establish synergies between the concurrency and programming language communities working on the theory and practice of SOS. Moreover, it aims at widening the knowledge of SOS among postgraduate students and young researchers worldwide.

The First SOS Workshop took place on 30th August 2004 in London (UK) as a satellite event of CONCUR 2004, the *Fifteenth International Conference on Concurrency Theory*, and marked the publication of two special volumes (60-61) of the *Journal of Logic and Algebraic Programming* devoted to SOS. The second SOS Workshop took place on 10th July 2005 in Lisbon (Portugal) as a satellite event of ICALP 2005, the *The 32nd International Colloquium on Automata, Languages and Programming*.

*This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science*

**Programme committee:**

- Rocco De Nicola (Florence, IT)
- Wan Fokkink (Amsterdam, NL)
- Rob van Glabbeek (NICTA, AU, co-chair)
- Reiko Heckel (Leicester, UK)
- Matthew Hennessy (Sussex, UK)
- Ugo Montanari (Pisa, IT)
- Peter Mosses (Swansea, UK, co-chair)
- MohammadReza Mousavi (Eindhoven, NL)
- David Sands (Chalmers, SE)
- Irek Ulidowski (Leicester, UK)
- Shoji Yuen (Nagoya, JP)

The submitted papers were refereed by the programme committee and by several outside referees, whose help is gratefully acknowledged.

**Invited speakers:**

- Bartek Klin (Warsaw, PL)
- Robin Milner (Cambridge, UK), joint Express-Infinity-SOS invited speaker

We are especially grateful to them for accepting the invitation to speak at the workshop, and for the papers that they have written for this occasion.

**Publication:**

The final versions of the papers in these preliminary proceedings will be published in ENTCS, *Electronic Notes in Theoretical Computer Science*. The proceedings of SOS 2004 appeared as ENTCS volume 128, issue 1. The proceedings of SOS 2005 appeared as ENTCS volume 156, issue 1; a special issue of *Theoretical Computer Science* based on selected papers is in preparation. ENTCS is published electronically through the facilities of Elsevier Science B.V. and under its auspices. We are grateful to ENTCS for their continuing support, and in particular to Mike Mislove, Managing Editor of the series.

**Organization:**

We are grateful to the CONCUR 2006 organizers for taking care of the local organization, and for managing the printing of these preliminary proceedings. Support from National ICT Australia and Swansea University is also gratefully acknowledged.

*Rob van Glabbeek (National ICT Australia)* *27th July 2006*
*Peter D. Mosses (Swansea University)*

# Bialgebraic Methods in Structural Operational Semantics

## (Invited Talk)

Bartek Klin [1,2]

*Warsaw University, Edinburgh University*

**Abstract**

Bialgebraic semantics, invented a decade ago by Turi and Plotkin, is an approach to formal reasoning about well-behaved structural operational specifications. An extension of algebraic and coalgebraic methods, it abstracts from concrete notions of syntax and system behaviour, thus treating various kinds of operational descriptions in a uniform fashion.

In this talk, the current state of the art in the area of bialgebraic semantics is presented, and its prospects for the future are sketched. In particular, a combination of basic bialgebraic techniques with a categorical approach to modal logic is described, as an abstract approach to proving compositionality by decomposing modal logics over structural operational specifications.

*Key words:* Structural operational semantics, category theory, algebra, coalgebra, bialgebra

## 1 Introduction

Since its invention in the early 1980's, Structural Operational Semantics (SOS) [36,35,1] has been one of the most popular and successful frameworks for the formal description of programming languages and process calculi. Not only it has become the formalism of choice for a clear and concise presentation of innumerable ideas and formalisms (see [5] for many examples), it has also been proved a viable option for the description of fully grown programming languages [31]. In the structural operational approach, the semantics of programs (or processes) is described by means of transition systems, induced by inference rules following their syntactic structure. The intuitive appeal of this

[1] Supported by EPSRC grant EP/D039045/1.

[2] Email: bklin@inf.ed.ac.uk

approach and, importantly, its inherent support for modelling nondeterministic behaviour, makes it a natural framework for describing computational languages.

There are many types of standard reasoning applied to structural operational descriptions to check whether they "behave well". For example, one often wants to prove that a certain equivalence relation on processes, defined by the transition system induced from a specification, is compositional and thus allows for inductive reasoning. Further, one might be interested in generating a set of equations characterising a chosen process equivalence. In other cases, a language with an operationally defined semantics is extended with new operators, and one would like to ensure that the extension is conservative, i.e., that the behaviour of the old operators is left unchanged. More generally, one would like to compare two languages and check whether one can be translated to the other such that its operational semantics, or a process equivalence, is preserved. One could also try to combine two languages, or a set of separately defined operators, and reason about the resulting language in a modular fashion, based on features of its components.

Proofs of properties such as listed above can be quite demanding, and it would be unfortunate if they had to be done from scratch for each language considered. Indeed, the multitude of existing examples of structural operational descriptions, and the continuous appearance of new ones, calls for a mathematical framework — a theory of SOS — that would facilitate standard types of reasoning performed on language specifications.

In the simplest and most widely studied form of SOS, operational specifications induce labelled transition systems (LTSs) where processes are closed terms over some algebraic signature, and labels have no specific structure. In that case, a reasonably general theory based on the notion of Transition System Specification (TSS) has been developed, and much progress towards overcoming the above difficulties has been made (see [1,14] for a survey). For most classical process equivalences congruence formats have been provided, i.e., syntactic restrictions on inference rules that guarantee the equivalences compositional. Much work on equational definability and conservative extensions has also been done. Least progress has been made in the area of language translation and modularity, with the notable exception of Modular Structural Operational Semantics by Mosses [32], which enjoys better modularity than its classical version when applied to standard cases.

However, the issue becomes more complex with the continuing appearance of new syntactic and computational paradigms dictated by the everchanging world of computing. Examples include probabilistic, timed and hybrid systems, ones with global or local state, name passing, process passing, and stochastic systems. All these features can be described in some forms of structural operational semantics. Arguably though, the treatment of these forms in the classical TSS framework is rather superficial, as it does not take into account the semantic structure of transition systems involved. As a result,

the existing approaches to checking whether semantic descriptions are well behaved, are hard to adapt to these new paradigms; instead one usually needs to rework the old solutions from scratch in each case.

It is therefore desirable to have a theory of SOS that would abstract from the syntactic and computational paradigms involved, and allows one to define and prove useful properties of semantic descriptions on an abstract level. With such a theory, SOS will hopefully be better prepared for future changes in the world of computing.

The basics of such a theory were proposed by Turi and Plotkin in [40]. Its beginnings lie in the coalgebraic account of transition systems (see [17] for a gentle introduction and [38] is a good reference), where the notion used to classify various kinds of processes is that of behaviour, modelled as a functor on a category and formally representing the vague concept of computational paradigm. Combined with the classical algebraic approach to syntax, these techniques lead to the development of *bialgebraic semantics* of processes, which turned out to generalise and explain many aspects of well-behaved operational semantics.

Since then, the bialgebraic approach has developed considerably, and it has lead to several results interesting to a wider community. The purpose of this extended abstract is to give a gentle introduction to basic bialgebraic techniques, aimed at researchers familiar with SOS; to sketch their current stage of development; and to present the author's personal views on their future prospects. Note that much of interesting work on bialgebras in semantics, some of it less directly related to SOS, has been omitted in this introductory paper, and the bibliography included is far from complete.

**Acknowledgements.** The author is grateful to Gordon Plotkin for continuing cooperation, and to Alexander Kurz for useful discussions.

## 2 Bialgebraic semantics

In this section, the framework of bialgebraic semantics is presented together with some results that have been obtained through its use. To appreciate the following development, the reader is expected to be familiar with the basics definitions and techniques of structural operational semantics (see [1] for a reference). Bialgebras are defined in the language of category theory, so familiarity with basic notions such as category, functor and natural transformation is also recommended (the first chapters of [2,30] are good references).

### *2.1 Processes as coalgebras*

In the standard framework of SOS as used e.g. in process algebra [1], structural operational descriptions specify labelled transition systems (LTSs), i.e., triples $\langle X, A, \rightarrow \rangle$, where $X$ is a set of processes, $A$ a set of labels, and $\rightarrow \subseteq X \times A \times X$

is a labelled transition relation. It is easy to see LTSs as functions

$$h : X \to \mathcal{P}(A \times X)$$

where $\mathcal{P}$ is the powerset construction and $\times$ is cartesian product. Indeed, an LTS maps a process $x \in X$ to the set of all tuples $\langle a, y\rangle$ such that $x \xrightarrow{a} y$. In the language of category theory, a map as above is called a *coalgebra* for the *functor* $\mathcal{P}(A \times -)$. In general, for any functor $B$, a $B$-coalgebra is a map (function) $h : X \to BX$ for some object (set) $X$.

As it happens, coalgebras for some other functors on the category **Set** of sets and functions correspond to other well-known types of transition systems. For example:

- Coalgebras for $\mathcal{P}_{\mathrm{f}}(A \times -)$, where $\mathcal{P}_{\mathrm{f}}$ is the *finite* powerset functor, are *finitely branching* LTSs. Coalgebras for $(\mathcal{P}_{\mathrm{f}}(-))^A$ are *image finite* LTSs.
- Coalgebras for $\mathcal{D}(A \times -)+1$, where $D$ is the probability distribution functor, are generative probabilistic transition systems.
- Coalgebras for $(S \times (1 + -))^S$ are deterministic transition systems with state and termination.

Many other examples of systems modelled as coalgebras for functors on **Set** can be found in [38]. Coalgebras for functors on other categories have also been used; for example, in [8], coalgebras for a certain functor on the category **NSet** of nominal sets and equivariant functions [13] are shown to correspond to a kind of labelled transition systems with name binding. The coalgebraic abstraction allows one to treat many different kinds of systems in a uniform manner. At the same time, many important notions used in reasoning about transition systems can be explained at the abstract, coalgebraic level; examples include canonical process equivalences such as bisimilarity, or modal logics such as the Hennessy-Milner logic.

### *2.2 Terms as algebras*

In simple types of structural operational semantics, processes are closed terms over some algebraic signature. It is standard to consider sets of such terms as algebras for certain functors on **Set**. For example, a language syntax described by the grammar

$$t ::= \mathtt{nil} \mid a.t \mid t + t \mid t||t\,,$$

where $a$ ranges over a fixed set $A$, corresponds to the functor

$$\Sigma X = 1 + A \times X + X \times X + X \times X$$

where 1 is a singleton set, $\times$ is cartesian product and $+$ is disjoint union. Note that an element of the set $\Sigma X$ can be seen as a simple term of the above grammar, build of exactly one syntactic construct with variables from $X$. It turns out that algebras for the signature (in the usual sense of universal

algebra) are maps

$$g : \Sigma X \to X$$

with $X$ the algebra carrier. This way, a simple syntax corresponds to a functor on **Set**. To model more advances syntactic features such as variable binding, one needs to move to more complex categories, such as **NSet**.

If a functor $\Sigma$ corresponds to an algebraic signature, then the set of terms over the signature and over a set $X$ of variables is denoted $T_\Sigma X$, or $TX$ if $\Sigma$ is clear from context. In particular, $T0$ is the set of closed terms over $\Sigma$. This set admits an obvious and canonical algebra structure, denoted $\psi : \Sigma T0 \to T0$. This $\Sigma$-algebra is *initial*: for any other algebra $g : \Sigma X \to X$ there is a unique homomorphism from $\psi$ to $g$, i.e., a map $g^\sharp : T0 \to X$ such that $g^\sharp \circ \psi = g \circ \Sigma g^\sharp$. Intuitively, $g^\sharp$ is defined by structural induction, where $g$ defines the inductive step. The construction $T$ is also a functor, and it is called the monad freely generated by $\Sigma$. The notions of initial algebra and freely generated monad does not depend on $\Sigma$ corresponding to an algebraic signature, and can be defined for many other functors. For more intuitions about this categorical approach to induction, see e.g. [17].

### *2.3 Abstract GSOS*

Simple structural operational descriptions induce LTSs with closed terms as processes. In other words, the set of processes is equipped with both a coalgebraic structure, which maps a process to a structure of its successors, and an algebraic structure, which describes how to obtain a process by combining other processes. Formally, induced LTSs are coalgebras $h : T0 \to BT0$ for a suitable behaviour $B$, and for $T$ the monad freely generated by syntax $\Sigma$.

To model the process of inducing LTSs with syntax abstractly, a sufficiently abstract notion of structural operational description is needed. For a first attempt, consider a standard set of operational inference rules for a toy language with synchronous product:

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \otimes y \xrightarrow{a} x' \otimes y'} \qquad \frac{}{a \xrightarrow{a} \mathtt{nil}} \tag{1}$$

where $a$ ranges over a fixed set $A$ of labels. The syntax of this language corresponds, as in §2.2, to the functor

$$\Sigma X = 1 + A + X \times X\,.$$

Rules (1) induce a standard LTS labelled with $A$, i.e., a coalgebra for the functor

$$BX = \mathcal{P}(A \times X)\,.$$

But how to model the rules on the abstract level? Informally, they define the behaviour (i.e., the set of successors) of a term built of a single syntactic construct and variables, based on some information about the behaviour of

subterms represented by the variables. For example, given processes $x, y$ from any set $X$, and sets of successors for $x$ and for $y$, the leftmost rule defines the set of successors for the process $x \otimes y$. Note that while successors of $x$ and $y$ are variables and therefore can be thought of as arbitrary elements of $X$, the derived successors of are simple terms from $\Sigma X$. Formally, the left rule in (1) can be modelled as a function

$$\lambda_{\otimes} : BX \times BX \to B\Sigma X$$

defined by

$$\lambda_{\otimes}(\beta, \gamma) = \{\, \langle a, x \otimes y\rangle \in A \times \Sigma X \mid \langle a, x\rangle \in \beta \wedge \langle a, y\rangle \in \gamma \,\} \,.$$

Similarly, the right rule represents a function $\lambda_A : A \to B\Sigma X$ defined by

$$\lambda_A(a) = \{\langle a, \mathtt{nil}\rangle\} \,,$$

and even the lack of any rules for the construct `nil` defines its behaviour: the process `nil` has no successors. This can be viewed as a function $\lambda_{\mathtt{nil}} : 1 \to B\Sigma X$:

$$\lambda_{\mathtt{nil}}(*) = \emptyset \,.$$

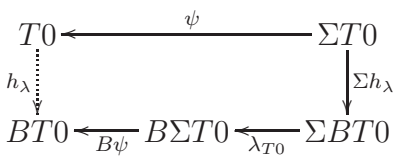
The three functions can be combined into a function

$$\lambda : \Sigma BX \to B\Sigma X$$

defined by cases and corresponding to (1). Note that the structure of $X$ and the nature of its elements are completely ignored in the definition of $\lambda$. Formally, $\lambda$ is natural over $X$:

$$\lambda : \Sigma B \Longrightarrow B\Sigma \,. \tag{2}$$

A natural transformation like this is called a *distributive law* of $\Sigma$ over $B$, and a first attempt to model structural operational rules would be to consider distributive laws of the syntax functor over the behaviour functor. We have just seen a reasonable example covered by this notion. Moreover, it turns out that the process of inferring LTSs from SOS rules can be explained abstractly at the level of distributive laws. Indeed, the unique algebra morphism $h_\lambda$ from the initial $\Sigma$-algebra as below:




is an LTS of the required type.[3] The reader is encouraged to check that for

[3] The pair $\langle \psi, h_\lambda\rangle$ is a $\lambda$-*bialgebra* [40], the central notion of bialgebraic semantics.

the transformation $\lambda$ defined as above, the inductively defined $h_\lambda$ is exactly the expected LTS induced by (1).

The above encourages one to model sets of SOS rules as distributive laws. However, there are many examples which do not fit into the simple framework described so far. Consider, for example, rules like

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{x}+\mathtt{y} \xrightarrow{a} \mathtt{x}'} \qquad \frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{x}|\mathtt{y} \xrightarrow{a} \mathtt{x}'|\mathtt{y}} \qquad \frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{!\mathtt{x} \xrightarrow{a} \mathtt{x}'|!\mathtt{x}}\,. \tag{3}$$

According to the first rule, an inferred successor of a process $x + y$ does not need to be a term built of a single syntactic construct; indeed, it is merely a variable. In the second rule, the inferred successor is not built solely of successors of the subprocesses $x$ and $y$; instead, the variable $y$ is used itself. In the third rule, both problems occur, although here the inferred successor is not a variable, but a complex term with more than one syntactic construct. All these rules cannot be represented as a natural transformation (2) for the obvious choices of $B$ and $\Sigma$.

A more general framework was proposed already in the original paper [40], where distributive laws

$$\lambda : \Sigma(\mathrm{Id} \times B) \Longrightarrow BT \tag{4}$$

were considered. $B$-coalgebras can be induced from such laws much the same as shown above for the simple laws. It turns out that for $B = \mathcal{P}(A \times -)$, and for $\Sigma$ and $T$ corresponding to an algebraic signature, laws of this kind correspond exactly to specifications in the well-known format GSOS [6]; hence the name *abstract GSOS*. In particular, the three problematic rules (3) can be modelled as distributive laws.

In [40], distributive laws dual to (4) were also considered, i.e., natural transformations

$$\nu : \Sigma D \Longrightarrow B(\mathrm{Id} + \Sigma) \tag{5}$$

where $D$ is the comonad cofreely generated by $B$, just as $T$ in (4) is the monad freely generated by $\Sigma$. For $B = \mathcal{P}_\mathrm{f}(A \times -)$, $DX$ is the set of finite or infinite trees edge-labelled by $A$ and node-labelled by $X$, quotiented by strong bisimilarity, and distributive laws (5) correspond to sets of SOS rules in the *safe ntree format* [11,40].

Both GSOS and safe ntree formats guarantee bisimilarity to be a congruence on the induced LTS. An important contribution of [40] was to show that these congruence properties can be proved on the abstract level of distributive laws, and thus they are immediately translated to SOS frameworks based on different notions of syntax and behaviour. Several applications of this result, together with some other work on bialgebraic semantics published so far, are mentioned in the remainder of this section.

### *2.4 Categorical foundations*

In [40], natural transformations of the type (4) and (5) are considered as special cases of the more general notion of a distributive law of a monad over a comonad. In [28,29,37], various types of distributive laws are studied on the abstract, categorical level. In [4], different kinds of distributive laws are studied and related on the concrete example of LTSs; also a complete proof of one-to-one correspondence between abstract GSOS and concrete GSOS specifications is included there.

### *2.5 Abstract GSOS for probabilistic and timed systems*

In [3,4], the abstract GSOS framework is applied to reactive probabilistic systems and probabilistic automata, represented as coalgebras for suitable functors. A congruence format for probabilistic bisimilarity is derived.

In [18,19], the same framework is applied to processes with timed transitions. Congruence results regarding time bisimilarity are proved, and a congruence format for the case of discrete time is derived. In [20,21], the combination of timing with action is studied more carefully, with insights on combining different behaviours to obtain a modular account of semantics.

### *2.6 Recursion*

In [23], abstract GSOS is studied in a CPO-enriched setting, where recursion is possible to express via straightforward fixpoint constructions. There, it is shown how to combine standard GSOS distributive laws with recursive equations to obtain other well-behaved distributive laws. Another bialgebraic approach to recursive equations is [16].

### *2.7 Name binding*

In [9,10], syntax with variable binding was modelled algebraically in a presheaf category, and the standard SOS description of the $\pi$-calculus was shown to fit in the abstract GSOS format there, although no actual format was proposed. Recently [8], such a format, a special case of abstract GSOS, has been proposed in the closely related setting of nominal sets [13], with congruence properties related to a version of open bisimilarity. Interestingly, in nominal settings the syntax and behaviour functors reside in different categories. The basic bialgebraic setting is suitably generalised to accommodate this.

### *2.8 Van Glabbeek spectrum*

In [25,22,24], abstract GSOS is interpreted in certain fibered categories. This allows one to derive congruence formats for process equivalences other than the canonical coalgebraic notion of bisimilarity. In particular, novel formats for completed trace and failures equivalences on LTSs were obtained.

## 3 Future directions

Bialgebraic techniques are still in the initial stage of development and much remains to be done if a general and practical theory of structural operational semantics is to be achieved. This section briefly presents the author's personal view on the most promising directions of development, and the most important challenges to be taken.

### *3.1 Relations to reactive systems*

In modern process algebra, much attention is paid to a semantic framework alternative to SOS, i.e., to reactive systems [27]. Many languages, such as the $\pi$-calculus or ambient calculus, are naturally described in the language of reactive systems, where dynamic behaviour is described by an unlabelled reaction relation together with a suitable structural congruence. The reactive approach is quite intuitive and easy to use; however, it imposes less structure on the described language and in particular it does not easily facilitate compositionality. It would be very desirable to build a bridge between SOS and reactive systems, and be able to translate or compare operational descriptions between the two formalisms. It seems that sufficiently abstract theories of both approaches is indispensable to that end. The bialgebraic approach will hopefully become such a theory for SOS, and [27,39] can be seen as attempts to develop such a theory for reactive systems.

### *3.2 Modal logic decomposition*

In a coalgebraic approach to modal logics and system testing [15,26,34], one considers a contravariant adjunction between a category $\mathcal{C}$ of processes, where coalgebras for a functor $B$ are systems, and a category $\mathcal{D}$ of tests, with a functor $L$ representing the syntax of a logic. The functors $B$ and $L$ are connected along the adjunction, and the connection provides an interpretation of the logic over $B$-coalgebras.

In the bialgebraic setting, processes are equipped with syntax represented by a functor $\Sigma$ on $\mathcal{C}$, with a distributive law $\lambda$ of $\Sigma$ over $B$ representing operational semantics. It is then natural to consider a functor $\Gamma$ on $\mathcal{D}$, connected to $\Sigma$ along the adjunction and representing the "behaviour of the logic". Intuitively, $\Gamma$-coalgebras describe ways of decomposing modal logics over the syntax $\Sigma$. One can try to come up with a distributive law of $L$ over $\Gamma$, connected to $\lambda$ along the adjunction. If this succeeds, then logic decomposition is compatible with the operational semantics and the process equivalence defined by the logic $L$ is compositional with respect to the language defined by $\lambda$. This provides a general framework for proving congruence results, of which [22,24] is a special case. On the concrete level, the resulting congruence principle is related to work on decomposing modal logics such as [12].

### *3.3 Equations and weak equivalences*

An important tool in practical operational semantics are equations between process terms, allowing language designers to prepare shorter and more intuitive descriptions. While in principle, equations are easily to express on the abstract bialgebraic level by considering syntactic monads other than the freely generated ones, the development of concrete formats based on this observation is a difficult challenge. One would like to match the more concrete development of [33].

Dually, it is also important to treat weak equivalences such as weak bisimilarity in the bialgebraic setting, providing congruence results for them. This would require a general coalgebraic approach to weak equivalences, a major challenge with no satisfactory solution so far.

### *3.4 Modularity*

The ultimate practical goal of a theory of SOS must be a general, easy-to-use and expressive framework for the modular development of operational semantics. The ambition of the bialgebraic approach is to make the framework parametrised by notions of syntax and behaviour. However, many problems need to be solved before such a framework appears. A theory of well-behaved translations between operational descriptions needs to be developed, and notions of morphisms between distributive laws such as those in [41] seem good steps to this end. Modular construction of SOS descriptions will also require techniques for combining rules based on different types of behaviour. Initial attempts to that end were made in [20,21]. In the coalgebraic world, much work on composing behaviours has been done in relation with modal logics [7].

## References

[1] L. Aceto, W. J. Fokkink, and C. Verhoef. Structural operational semantics. In J. A. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*. Elsevier, 2002.

[2] J. Adámek, H. Herrlich, and G. E. Strecker. *Abstract and Concrete Categories*. 2004. Available from `http://katmat.math.uni-bremen.de/acc`.

[3] F. Bartels. GSOS for probabilistic transition systems. In *Proc. CMCS'02*, volume 65 of *ENTCS*, 2002.

[4] F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. PhD dissertation, CWI, Amsterdam, 2004.

[5] J. A. Bergstra, A. Ponse, and S. Smolka. *Handbook of Process Algebra*. Elsevier, 2002.

[6] B. Bloom, S. Istrail, and A. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42:232–268, 1995.

[7] C. Cirstea and D. Pattinson. Modular construction of modal logic. In *Proc. CONCUR'04*, volume 3170 of *Lecture Notes in Computer Science*, 2004.

[8] M. Fiore and S. Staton. A congruence rule format for name-passing process calculi from mathematical structural operational semantics. In *Proc. LICS'06*. IEEE Computer Society Press, 2006.

[9] M. P. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax with variable binding. In *Proc. LICS'99*, pages 193–202. IEEE Computer Society Press, 1999.

[10] M. P. Fiore and D. Turi. Semantics of name and value passing. In *Proc. LICS'01*, pages 93–104. IEEE Computer Society Press, 2001.

[11] W. Fokkink and R. van Glabbeek. Ntyft/ntyxt rules reduce to ntree rules. *Information and Computation*, 126, 1996.

[12] W. J. Fokkink, R. J. van Glabbeek, and P. de Wind. Compositionality of Hennessy-Milner logic through structural operational semantics. In *Proc. FCT'03*, volume 2751 of *LNCS*, 2003.

[13] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2001.

[14] J. F. Groote, M. Mousavi, and M. A. Reniers. A hierarchy of sos rule formats. In *Proc. SOS'05*.

[15] B. Jacobs. Towards a duality result in the modal logic for coalgebras. In *Proc. CMCS 2000*, volume 33 of *ENTCS*, 2000.

[16] B. Jacobs. Distributive laws for the coinductive solution of recursive equations. *Information and Computation*, 204, 2006.

[17] B. Jacobs and J. J. M. M. Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the EATCS*, 62, 1996.

[18] M. Kick. Bialgebraic modelling of timed processes. In *Proc. ICALP'02*, volume 2380 of *LNCS*, 2002.

[19] M. Kick. Rule formats for timed processes. In *Proc. CMCIM'02*, volume 68 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.

[20] M. Kick and J. Power. Modularity of behaviours for mathematical operational semantics. In *Procs. CMCS'04*, volume 106 of *ENTCS*, 2004.

[21] M. Kick, J. Power, and A. Simpson. Coalgebraic semantics for timed processes. *Information and Computation*. To appear.

[22] B. Klin. *Abstract Coalgebraic Approach to Process Equivalence for Well-Behaved Operational Semantics*. PhD thesis, BRICS, Aarhus University, 2004.

[23] B. Klin. Adding recursive constructs to bialgebraic semantics. *Journal of Logic and Algebraic Programming*, 60-61, 2004.

[24] B. Klin. From bialgebraic semantics to congruence formats. In *Proc. SOS 2004*, volume 128 of *ENTCS*, 2005.

[25] B. Klin and P. Sobocinski. Syntactic formats for free: An abstract approach to process equivalence. In *Proc. CONCUR 2003*, volume 2671 of *LNCS*, 2003.

[26] A. Kurz. Coalgebras and their logics. *ACM SIGACT News*, 37, 2006.

[27] J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In *Proc. CONCUR 2000*, volume 1877 of *LNCS*, 2000.

[28] M. Lenisa, J. Power, and H. Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. In *Proc. CMCS'00*, volume 33 of *Electronic Notes for Theoretical Computer Science*. Elsevier, 2000.

[29] M. Lenisa, J. Power, and H. Watanabe. Category theory for operational semantics. *Theor. Comput. Sci.*, 327(1-2), 2004.

[30] S. Mac Lane. *Categories for the Working Matematician*. Springer-Verlag, second edition, 1998.

[31] R. Milner and M. Tofte. *The definition of Standard ML*. MIT Press, revised edition, 1997.

[32] P. D. Mosses. Modular structural operational semantics. *Journal of Logic and Algebraic Programming*, 60-61, 2004.

[33] M. R. Mousavi and M. A. Reniers. Congruence for structural congruences. In *Proc. FOSSACS'05*, volume 3441 of *LNCS*, 2005.

[34] D. Pavlovic, M. Mislove, and J. B. Worrell. Testing semantics: connecting processes and process logics. In *Proc. AMAST'05*, volume 4019 of *LNCS*, 2005.

[35] G. D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming*, 60-61, 2004.

[36] G. D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.

[37] J. Power and H. Watanabe. Distributivity for a monad and a comonad. In *Procs. CMCS'99*, volume 19 of *Electronic Notes on Theoretical Computer Science*. Elsevier Science Publishers, 1999.

[38] J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.

[39] V. Sassone and P. Sobocinski. Locating reaction with 2-categories. *Theoretical Computer Science*, 333, 2003.

[40] D. Turi and G. D. Plotkin. Towards a mathematical operational semantics. In *Proc. LICS'97*, pages 280–291. IEEE Computer Society Press, 1997.

[41] H. Watanabe. Well-behaved translations between structural operational semantics. *Electronic Notes in Theoretical Computer Science*, 65, 2002.

# On Well-Foundedness and Expressiveness of Promoted Tyft

## (Being Promoted Makes a Difference)

MohammadReza Mousavi [1], Michel Reniers

*Department of Computer Science, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands*

**Abstract**

In this paper, we solve two open problems posed by Karen L. Bernstein regarding her promoted tyft format for structured operational semantics. We show that, unlike formats with closed terms as labels, such as the tyft format, the well-foundedness assumption cannot be dropped for the promoted tyft format while preserving the congruence result. We also show that the well-founded promoted tyft format is incomparable to the tyft format with closed terms as labels, i.e., there are transition relations that can be specified by the promoted tyft format but not by the tyft format, and vice versa.

*Key words:* Structural Operational Semantics (SOS), SOS Rule Formats, Promoted Tyft, Tyft.

## 1 Introduction

In [1], Bernstein proposed the promoted tyft format which is an elegant framework for specifying the operational semantics of higher-order processes. She proved that the *well-founded* promoted tyft format guarantees strong bisimilarity to be a congruence. The conclusions of [1] reads as follows.

> "In this paper, we have described a rule format that is a simple but expressive generalization of Groote and Vaandrager's tyft/tyxt rule format. ... There are several open questions related to the work in this paper. It is not clear that the well-foundedness property is necessary for the congruence result. We are not sure how the extensions to tyft/tyxt format that allow negative premises are compatible with our extensions. It is not clear whether promoted tyft/tyxt format is strictly more expressive than tyft/tyxt format."

[1] Corresponding author: MohammadReza Mousavi, m.r.mousavi@tue.nl

*This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science*

We touched upon the second open question in another publication [2]. In this paper, we answer the first and the third questions as follows.

- We show that the promoted tyft format does not necessarily induce congruence of strong bisimilarity if the well-foundedness assumption is omitted;
- We show that the well-founded subset of the promoted tyft format is incomparable, in its expressiveness, with the tyft format. In other words, we give two counter-examples witnessing that there exist transition relations that can be specified by one rule format but not by the other.

The rest of this paper is organized as follows. In Section 2, we give some basic definitions. Section 3 addresses the well-foundedness concept, shows that it cannot be dropped for the promoted tyft format while preserving the congruence result. Section 4 addresses the expressiveness of the promoted tyft format and proves it incomparable to the tyft format. The paper is concluded in Section 5.

## 2 Preliminaries

**Definition 2.1 (Signature, Term and Substitution)** *Assume a countable set of variables $V$ (with typical members $x$, $y$, $x'$, $y'$, $x_i$, $y_i \ldots$). A* signature *$\Sigma$ is a set of function symbols (operators, with typical members $f$, $g$, $\ldots$) with fixed arities $ar : \Sigma \to \mathbb{N}$. Functions with zero arity are called constants. Terms $s, t, t_i, \ldots \in \mathcal{T}$ are constructed inductively using variables and function symbols. A list of terms is denoted by $\overrightarrow{t}$. When we write $f(\overrightarrow{t})$, we assume that $\overrightarrow{t}$ has the right size, i.e., $ar(f)$. All terms are considered open terms. Closed terms $p, q, \ldots \in \mathcal{C}$ are terms that do not mention a variable and are typically denoted by $p, q, p', q', p_i, \ldots$. A substitution $\sigma$ replaces variables in a term with terms. The set of variables appearing in term $t$ is denoted by $vars(t)$.*

**Definition 2.2 (Transition System Specification (TSS))** *A* transition system specification *is a tuple $(\Sigma, D)$ where $\Sigma$ is a signature and $D$ is a set of deduction rules. A deduction rule $dr \in D$, is defined as a tuple $(H, c)$ where $H$ is a set of formulae and $c$ is a formula. For all $t, t', t'' \in \mathcal{T}$ we define that $t \xrightarrow{t'} t''$ is a formula. The formula $c$ is called the* conclusion *of $dr$ and the formulae from $H$ are called its* premises. *A deduction rule $(H, c)$ is mostly denoted by $\frac{H}{c}$.*

*The concept of substitution is lifted to formulae and sets of formulae in the natural way (i.e., a substitution applied to a formula, applies to all three terms). We refer to $t$ as the source, $t'$ as the label and $t''$ as the target of the transition. We may also write $vars(\phi)$ and $vars(H)$ to denote variables appearing in a formula and in a set of formulae, respectively.*

**Definition 2.3 (Tyft [3] and Promoted Tyft [1])** *A deduction rule is in*

*tyft format if and only if it has the following form*

$$\frac{\{t_i \xrightarrow{t'_i} y_i | i \in I\}}{f(\overrightarrow{x}) \xrightarrow{t'} t''},$$

*where variables in $\overrightarrow{x}$ and $y_i$'s are all distinct variables, all labels, i.e., $t'$ and $t'_i$'s, are closed terms, $I$ is a (possibly infinite) set of indices.*

*A rule of the above form is in the promoted tyft format if the source and targets of all formulae in it conform to the constraints of the tyft format and further $t'_i$'s contain at least one function symbol (i.e., are not variable), $t'$ is of the form $g(\overrightarrow{z})$ where variables in $\overrightarrow{z}$ are all distinct and different from variables in $\overrightarrow{x}$ and $y_i$'s.*

*A transition system specification is in tyft (promoted tyft) format if and only if all its deduction rules are.*

A subset of the tyft format is the one using constants (instead of closed terms) as labels which is also considered in this paper and compared to the promoted tyft format in Section 4. Arguably, this subset can be considered the original definition of the tyft format as defined by [3]. The generalization to *closed* terms as labels (if at all considered a generalization) is entirely safe and orthogonal to all existing results (e.g., congruence, conservativity, commutativity meta-theorems).

The transition relation induced by a TSS (in the above two formats) is the set of all provable formulae as defined below.

**Definition 2.4** *A* proof *of a closed formula $\phi$ is a well-founded upwardly branching tree whose nodes are labelled by closed formulae such that*

- *the root node is labelled by $\phi$, and*
- *if $\psi$ is the label of a node and $\{\psi_i \mid i \in I\}$ is the set of labels of the nodes directly above this node, then there are a deduction rule $\frac{\{\chi_i \mid i \in I\}}{\chi}$ and a substitution $\sigma$ such that $\sigma(\chi) = \psi$, and for all $i \in I$, $\sigma(\chi_i) = \psi_i$.*

**Definition 2.5 (Strong (Bi)similarity)** *A relation $R \subseteq \mathcal{C} \times \mathcal{C}$ is a strong simulation relation when $\forall_{p,q \in \mathcal{C}}\ pRq \Rightarrow \forall_{p',p'' \in \mathcal{C}}\ p \xrightarrow{p'} p'' \Rightarrow \exists_{q'' \in \mathcal{C}}\ q \xrightarrow{p'} q'' \wedge (p'', q'') \in R$. A symmetric strong simulation relation is a strong bisimulation relation. Closed terms $p$ and $q$ are (bi)similar, denoted by $p \lesssim q$ ($p \leftrightarrows q$) if there is a strong (bi)simulation relation $R$ such that $p\ R\ q$.*

For a relation $R \subseteq \mathcal{C} \times \mathcal{C}$, we write $\overrightarrow{p}\ R\ \overrightarrow{q}$ and by that we mean $\overrightarrow{p}$ and $\overrightarrow{q}$ have the same size (possibly zero) and for all $p_i$ and $q_i$ at the same position in the two lists $p_i\ R\ q_i$.

**Lemma 2.6 (Substituting Bisimilar Labels Under Context [1])** *For a TSS in the promoted tyft format, $\forall_{p,q,\overrightarrow{p'},\overrightarrow{q'},p'' \in \mathcal{C}} \forall_{f \in \Sigma}\ p \xrightarrow{f(\overrightarrow{p'})} p'' \wedge p \leftrightarrows q \wedge \overrightarrow{p'} \leftrightarrows \overrightarrow{q'}$*

$\Rightarrow \exists_{q'' \in \mathcal{C}} q \xrightarrow{f(\overrightarrow{q'})} q'' \wedge p'' \leftrightarrows q''$.

**Definition 2.7 ((Pre-)Congruence)** *An equivalence (a pre-order) $R \subseteq \mathcal{C} \times \mathcal{C}$ is a* (pre-)congruence *when* $\forall_{f \in \Sigma} \forall_{\overrightarrow{p}, \overrightarrow{q} \in \mathcal{C}} \ \overrightarrow{p} \ R \ \overrightarrow{q} \Rightarrow f(\overrightarrow{p}) \ R \ f(\overrightarrow{q})$.

## 3 Well-Foundedness

In [1], Bernstein proposes a definition of well-foundedness which coincides with the following and proves that for the well-founded subset of the promoted tyft format, bisimilarity is a congruence.

**Definition 3.1** *The variable dependency graph of a deduction rule is a graph of which the nodes are variables and there is an edge from $x$ to $y$ when $y$ appears in the target of a premise and $x$ in its source or label. A deduction rule is well-founded when there is no backward chain of infinite length in the variable dependency graph. A TSS is well-founded when all its deduction rules are.*

Note that this definition coincides with that of [3] in case of TSS's with closed terms as labels. An alternative definition of well-foundedness is the one that treats the labels in the same way as the targets of formulae (while in the above definition labels are treated like sources). This alternative definition, called *p-well-foundedness* in [2], is not useful for proving congruence of strong bisimilarity (while it is useful for proving congruence of higher-order bisimilarity) and in fact, as shown below, there are s-well-founded TSS's in the promoted tyft format for which bisimilarity is not a congruence.

**Theorem 3.2 (Congruence for Well-founded (Promoted [1]) Tyft [3])** *For a well-founded TSS in the (*promoted*) tyft format, strong bisimilarity is a congruence.*

**Theorem 3.3 (Tyft Reduces to Well-founded Tyft [4])** *For an arbitrary TSS in the tyft format, there exists a well-founded TSS in the tyft format which induces the same transition relation.*

In the following three examples, we show that the congruence result for bisimilarity can be ruined if the transition system specifications in the promoted tyft format do not satisfy the well-foundedness assumption. The first example violates the well-foundedness assumption by having a self-loop on a variable which appears both in the label and the target of a premise.

**Example 3.4** *Consider the following set of deduction rules defined on a signature with* 0 *and* 1 *and $f$ as a unary function symbol.*[2] *The following TSS is in the promoted tyft format. Note that the last deduction rule is not well-founded due to the occurrence of $y$ both in the target and the label of the*

[2] In the coming examples we omit stating the precise signature as it is clear from the symbols used in the deduction rules.

*premise. (This deduction rule is indeed s-well-founded.)*

$$\frac{}{0 \xrightarrow{0} 1} \quad \frac{}{1 \xrightarrow{0} 0} \quad \frac{x \xrightarrow{0} y}{1 \xrightarrow{f(x)} x} \quad \frac{x \xrightarrow{0} y}{0 \xrightarrow{f(x)} y} \quad \frac{x \xrightarrow{f(y)} y}{f(x) \xrightarrow{1} y}$$

*The following is the transition relation induced by the above TSS.*

$$\{0 \xrightarrow{0} 1, \quad 1 \xrightarrow{0} 0, \quad 1 \xrightarrow{f(0)} 0, \quad 1 \xrightarrow{f(1)} 1, \quad 0 \xrightarrow{f(0)} 1, \quad 0 \xrightarrow{f(1)} 0, \quad f(1) \xrightarrow{1} 0, \quad f(1) \xrightarrow{1} 1\}$$

*Note that for the above transition relation it holds that* $0 \leftrightarrow 1$*, but it does not hold that* $f(0) \leftrightarrow f(1)$*. Therefore, bisimilarity is not a congruence.*

In the following two examples, the same exercise is repeated, i.e., it is shown that although the TSS is in the promoted tyft format and $0 \leftrightarrow 1$, it does not hold that $f(0) \leftrightarrow f(1)$. In the next example, the TSS is not well-founded since a variable in the target of a premise also appears in the source of the same premise and thus has a self-loop in the variable dependency graph.

**Example 3.5** *Consider the following TSS in the promoted tyft format. The last deduction rule is not well-founded.*

$$\frac{}{0 \xrightarrow{0} 0} \quad \frac{}{1 \xrightarrow{0} 0} \quad \frac{x \xrightarrow{0} y}{0 \xrightarrow{f(x)} x} \quad \frac{x \xrightarrow{0} y}{1 \xrightarrow{f(x)} y} \quad \frac{y \xrightarrow{f(x)} y}{f(x) \xrightarrow{1} y}$$

*The following is the transition relation induced by the above TSS.*

$$\{0 \xrightarrow{0} 0, \quad 1 \xrightarrow{0} 0, \quad 0 \xrightarrow{f(0)} 0, \quad 0 \xrightarrow{f(1)} 1, \quad 1 \xrightarrow{f(0)} 0, \quad 1 \xrightarrow{f(1)} 0, \quad f(0) \xrightarrow{1} 0\}$$

The last example violates well-foundedness (and congruence of bisimilarity) by having a non-trivial cycle concerning target, label and source of two premises.

**Example 3.6** *Consider the following TSS in the promoted tyft format. The last deduction rule is not well-founded.*

$$\frac{}{0 \xrightarrow{0} 1} \quad \frac{}{1 \xrightarrow{0} 0} \quad \frac{}{0 \xrightarrow{1} 0} \quad \frac{}{1 \xrightarrow{1} 1} \quad \frac{x \xrightarrow{0} y}{1 \xrightarrow{f(x)} x} \quad \frac{x \xrightarrow{0} y}{0 \xrightarrow{f(x)} y} \quad \frac{x \xrightarrow{f(y)} y' \quad y' \xrightarrow{1} y}{f(x) \xrightarrow{1} y}$$

*The following is the transition relation induced by the above TSS.*

$$\{0 \xrightarrow{0} 1, \quad 1 \xrightarrow{0} 0, \quad 0 \xrightarrow{1} 0, \quad 1 \xrightarrow{1} 1,$$
$$0 \xrightarrow{f(0)} 1, \quad 0 \xrightarrow{f(1)} 0, \quad 1 \xrightarrow{f(0)} 0, \quad 1 \xrightarrow{f(1)} 1, \quad f(1) \xrightarrow{1} 0, \quad f(1) \xrightarrow{1} 1\}$$

The essence of all counter-examples given before is the presence of a cycle in the variable dependency graph. Such cycles may allow for checking syntactic

equivalence of terms (e.g., comparing the argument in the target of a premise against a constant) and hence ruin the congruence result. An interesting question is whether there exists a subset of non-well-founded promoted tyft which indeed guarantees congruence, we conjecture that the *safe* subset of the promoted tyft format, as defined below, is the desired subset which guarantees congruence.

**Definition 3.7 (Safe Cycles)** *Consider a cycle $u_0 \to \ldots \to u_n \to u_0$ in the variable dependency graph of a deduction rule of the following form:*

$$\frac{\{t_i \xrightarrow{t'_i} y_i | i \in I\}}{f(\overrightarrow{x}) \xrightarrow{g(\overrightarrow{z})} t''},$$

*Such a cycle is called safe if in the variable dependency graph, there is no path $u \to \ldots \to u_i$ for all $i$, $0 \leq i \leq n$ such that $u$ is among $\overrightarrow{x}$ or among $\overrightarrow{z}$. A deduction rule (TSS) is safe when all cycles in its variable dependency graph (all its deduction rules) are safe.*

The following deduction rule contains a safe cycle in its premise.

$$\frac{c \xrightarrow{f(y)} y}{f(x) \xrightarrow{g(x)} y}$$

# 4 Expressiveness

## *4.1 Well-Founded* **Promoted Tyft** *does not reduce to* **Tyft**

Consider the following TSS in the promoted tyft format.

$$\frac{}{0 \xrightarrow{0} 0} \quad \frac{}{1 \xrightarrow{0} 0} \quad \frac{}{0 \xrightarrow{0} 1} \quad \frac{}{1 \xrightarrow{0} 1}$$

$$\frac{}{0 \xrightarrow{1} 0} \quad \frac{}{1 \xrightarrow{1} 0} \quad \frac{}{0 \xrightarrow{1} 1} \quad \frac{}{1 \xrightarrow{1} 1}$$

$$\frac{x \xrightarrow{0} y}{0 \xrightarrow{f(x)} 1} \quad \frac{x \xrightarrow{0} y}{1 \xrightarrow{f(x)} 0} \quad \frac{x \xrightarrow{f(x)} y}{0 \xrightarrow{f(x)} y} \quad \frac{x \xrightarrow{f(x)} y}{1 \xrightarrow{f(x)} y} \quad \frac{x \xrightarrow{f(x)} y}{f(x) \xrightarrow{1} y}$$

The transition relation induced by the above TSS is as follows.

$$\{0 \xrightarrow{0} 0;1\ ,\ 1 \xrightarrow{0} 0;1\ ,\ 0 \xrightarrow{1} 0;1\ ,\ 1 \xrightarrow{1} 0;1\ ,$$
$$0 \xrightarrow{f(0)} 1\ ,\ 0 \xrightarrow{f(1)} 0;1\ ,\ 1 \xrightarrow{f(0)} 0;1\ ,\ 1 \xrightarrow{f(1)} 0\ ,\ f(0) \xrightarrow{1} 1\ ,\ f(1) \xrightarrow{1} 0\}$$

where $p \xrightarrow{p'} p''; q''$ means $p \xrightarrow{p'} p''$ and $p \xrightarrow{p'} q''$. We claim that the above transition relation cannot be specified by any TSS in the tyft format.

If there is such a TSS, then there is a TSS in the pure well-founded tyft format which induces the same transition relation as above [4].

Consider the pure well-founded TSS in the tyft format that (purportedly) induces the same transition relation as above. Assume, without loss of generality that the proof of $f(0) \xrightarrow{1} 1$ from such a TSS does not depend on the proof for $f(1) \xrightarrow{1} 0$ (otherwise, a similar assumption should hold for the transition of $f(1) \xrightarrow{1} 0$ and one can swap 0's and 1's in the sources, labels and targets of the transitions in the remainder of the proof and the argument remain valid). The last deduction rule applied to derive the proof for $f(0) \xrightarrow{1} 1$ should be of the following form.

$$\textbf{(dr)}\frac{\{t_i \xrightarrow{p'_i} y_i | i \in I\}}{f(x) \xrightarrow{1} t''},$$

and there is a substitution $\sigma$ such that $\sigma(x) = 0$, $\sigma(t'') = 1$ and all $\sigma(t_i \xrightarrow{p'_i} y_i)$ have a proof tree.

**Definition 4.1 (Distance of a Variable)** *Given the above deduction rule, define the* distance *of variable $x$ as 0 and a variable $y_i$ to be the maximum of distances of variables appearing in $t_i$ plus 1. The distance of a premise is the distance of the variable of its target.*

Term $t''$ can either be a variable or the constant 1 (otherwise, if it contains a function symbol other than 1, $t''$ cannot be unified with 1). Since **(dr)** is pure, it can only contain variables $x$ or $y_i$'s ($i \in I$) and thus $t''$ can be either 1, or $x$ or $y_i$ (for some $i \in I$).

(i) If $t''$ is 1, i.e., if the deduction rule is of the following form

$$\frac{\{t_i \xrightarrow{p'_i} y_i | i \in I\}}{f(x) \xrightarrow{1} 1},$$

then we define substitutions $\sigma'_k$ inductively (on the rank of the premises) maintaining $\sigma(x) \leftrightarrow \sigma'_k(x)$ for all variables $x$ in the domain of $\sigma'_k$. First, define $\sigma'_0$ with $\sigma'_0(x) = 1$ and note that indeed $\sigma(x) = 0 \leftrightarrow 1 = \sigma'_0(x)$. Substitution $\sigma'_{k+1}$ is obtained from $\sigma'_k$ as follows: select a premise $t_i \xrightarrow{p'_i} y_i$ (or all such premises) for which the variables of the source are in the domain of $\sigma'_k$. Then, as $\sigma(t_i) \leftrightarrow \sigma'_k(t_i)$ (this follows from the fact that $\sigma(x) \leftrightarrow \sigma'_k(x)$ for all variables $x$ from the domain of $\sigma'_k$ and the fact that for a TSS in the tyft format, bisimilarity is a congruence) and $\sigma(t_i) \xrightarrow{p'_i} \sigma(y_i)$ we obtain the existence of $q'_i$ such that $\sigma'_k(t_i) \xrightarrow{p'_i} q'_i$ and $\sigma(y_i) \leftrightarrow q'_i$. Then define $\sigma'_{k+1}(y_i) \doteq q'_i$.

Define $\sigma'$ to be the supremum of the chain of premises $\sigma'_0, \sigma'_1, \ldots$ (which is increasing with respect to the subset ordering on their domains). Then, all premises of the deduction rule are derivable with respect to substitution $\sigma'$. Thus providing us with a proof for $f(1) \xrightarrow{1} 1$ (which is not supposed to be provable according to the above transition relation).

(ii) If $t''$ is $x$, then $\sigma(x) = \sigma(t'') = 0$ which is contradictory to the target of the transition $f(0) \xrightarrow{1} 1$.

(iii) Thus, it only remains to consider the case where $t''$ is a variable $y_c$, for some $c \in I$, i.e., the deduction rule is of the following form

$$\frac{\{t_i \xrightarrow{p'_i} y_i | i \in I\}}{f(x) \xrightarrow{1} y_c}.$$

Take an arbitrary variable $y_j$ such that $\sigma(y_j) = 1$ and define $\sigma'_0$ and $\sigma''_0$ to be the following partial substitutions:

$$\sigma'_0(x) = \sigma'_0(y_j) = 0 \qquad \text{and} \qquad \sigma''_0(x) = \sigma''_0(y_j) = 1.$$

Then, using an induction on the distance of $y_j$, we show that we can complete either $\sigma'_0$ or $\sigma''_0$ to a substitution $\sigma'$ such that the range of $\sigma'$ is $\{0, 1\}$ and for all $k \in I$, $\sigma'(t'_k \xrightarrow{p'_k} y_k)$ is provable.

Then, it follows that for the particular case of $y_c$, since $\sigma(y_c) = 1$, that we can prove either $f(0) \xrightarrow{1} 0$ or $f(1) \xrightarrow{1} 1$ which is contradictory to the transition relation that should be induced by the TSS.

- (Base case) If the distance of $y_j$ is 1, i.e., $y_j$ is the target of a premise of which the source only contains $x$ as variable or is a closed term, then the premise $t_j \xrightarrow{p'_j} y_j$ can be of one of the following eleven shapes (for all other transitions in the above transition relation, the target of the transition is 0 and thus cannot match with 1).

$$0; x \xrightarrow{0;1;f(0);f(1)} y_j \qquad \text{or} \qquad 1 \xrightarrow{0;1;f(0)} y_j,$$

where we have abused the ; notation to avoid writing all eleven cases explicitly.

For each of these eleven cases both substitutions $\sigma'_0$ and $\sigma''_0$ are complete. Furthermore, for each of the cases, at least one of these substitutions gives a transition that actually belongs to the transition relation induced by the TSS.

Assuming that $\sigma'_0$ is the substitution that proves the premise $t_j \xrightarrow{p'_j} y_j$, as before, one can complete the definition to a substitution $\sigma'$ inductively on the distance of the premises.

- (Induction step) Consider a rule in which $y_j$ has distance $n + 1$ for $n \geq 1$. As the distance of $y_j$ is $n + 1$, it cannot be the case that

$t_j$ is a closed term or the variable $x$, since then the distance of $y_j$ would have been 1. Hence, $t_j$ is a term containing at least a variable. Our previous assumption that the proof of $f(0) \xrightarrow{1} 1$ does not depend on a proof for $f(1) \xrightarrow{1} 0$ and the fact that all other transitions in the transition relation have a left-hand side 0 or 1 indicates that $t_j$ has to be a variable, say $y_k$. Now, suppose that $p'_j$ is 0, 1, or $f(1)$. Then, define the substitution $\sigma'$ to be $\sigma'(y_j) = 0$ and $\sigma'(v) = \sigma(v)$ for all other variables $v$ and this way we have a proof for all the premises using $\sigma'$ which is an extension of $\sigma'_0$. Thus it only remains to check the case where $p'_j = f(0)$. Therefore, the premise $t_j \xrightarrow{p'_j} y_j$ is of the form $y_k \xrightarrow{f(0)} y_j$ for some $k \in I$ where $y_k$ has distance $n$. Note that necessarily $\sigma(y_k) = 0$ since otherwise the substitution $\sigma'$ with $\sigma'(v) = \sigma(v)$ for all variables $v$ with distance smaller than the distance of $y_j$ and $\sigma'(y_j) = 0$ can be completed inductively on the rank of the premises to a substitution that extends $\sigma'_0$ and proves all the premises.

Based on a similar reasoning we must conclude that the premise $t_k \xrightarrow{p'_k} y_k$ should be of the form $y_l \xrightarrow{f(1)} y_k$ for some $l \in I$ where $y_l$ has distance $n-1$ and $\sigma(y_l) = 1$.

Thus we have a deduction rule of the following form:

$$\frac{y_l \xrightarrow{f(1)} y_k \qquad y_k \xrightarrow{f(0)} y_j \qquad \{t_i \xrightarrow{p'_i} y_i | i \in I - \{j,k\}\}}{f(x) \xrightarrow{1} y_c}.$$

By the induction step, we can complete the definition of one of the two following substitutions:

$$\sigma'_0(x) = \sigma'_0(y_l) = 0 \qquad \text{and} \qquad \sigma''_0(x) = \sigma''_0(y_l) = 1$$

to a substitution $\sigma'$ or $\sigma''$ such that all the premises with a distance of $n-1$ or less find a proof. If $\sigma'_0$ can be completed, then we define $\sigma'(y_k) = 1$ and $\sigma'(y_j) = 0$ and complete the definition of $\sigma'$ for all premises with distance $n$ or more, as before. If $\sigma''_0$ can be completed, we define $\sigma''(y_k) = 0$ and $\sigma''(y_j) = 1$ and complete the definition of $\sigma''$.

This concludes the proof as in all of the above cases, we can construct a proof for either $f(0) \xrightarrow{1} 0$ or $f(1) \xrightarrow{1} 1$ (or both) none of which are supposed to be in the induced transition relation.

### *4.2* **Tyft** *does not reduce to* **Promoted Tyft**

**Example 4.2** *Consider the following TSS in the* tyft *format. The signature of the TSS consists of* 0, 1 *and* 2 *as constants and* $f$ *as a unary function*

*symbol.*

$$\frac{}{2 \xrightarrow{f(0)} 2}$$

*The transition relation induced by it is* $\{2 \xrightarrow{f(0)} 2\}$*. We claim that there is no TSS in the* promoted tyft *format which can induce the same transition relation. It trivially holds that* $0 \leftrightarrows 1$ *and from* $2 \xrightarrow{f(0)} 2$ *and Lemma 2.6 that (for a TSS in the* promoted tyft *format)* $2 \xrightarrow{f(1)} 2$ *is also in the induced transition relation.*

If one restricts the tyft format to the subset with only constants as labels, then it trivially conforms to all requirements of the promoted tyft format and thus, the promoted tyft format (taking the first example in Section 4.1) is strictly more expressive than the tyft format with constants as labels.

### *4.3* (Promoted) Tyft *reduces to* Promoted PANTH

In [2], we introduced the promoted PANTH format which generalizes promoted tyft with negative premises. But even restricted to positive TSS's, the promoted PANTH format generalizes both the promoted tyft and the tyft format. To define the promoted PANTH format, we need the following notion of volatile operators.

**Definition 4.3** *(Volatile Operators) Given a TSS* $(\Sigma, D)$ *an operator* $f \in \Sigma$ *is called* volatile *when there exists a rule* $d \in D$ *of the following form:*

$$\frac{\{t_i \xrightarrow{t'_i} t''_i \mid i \in I\}}{t \xrightarrow{t'} t''}$$

*and* $f(\overrightarrow{t_k})$ *is a subterm of* $t'_i$ *for some* $i \in I$ *such that* $vars(\overrightarrow{t_k}) \cap vars(t) \neq \emptyset$ *or* $\exists_{i \in I} vars(\overrightarrow{t_k}) \cap vars(t'_i) \neq \emptyset$*.*

Note that for a TSS in the tyft format, no operator is volatile as the set $vars(\overrightarrow{t_k})$ is always empty.

The following is a simplified definition of the promoted PANTH format (restricted to positive TSS's and without predicates and lists of terms as labels) that suffices for our purposes.

**Definition 4.4** (Positive Promoted PANTH) *A deduction rule is in the* positive promoted PANTH *format when it is of the following form*

$$\frac{\{t_i \xrightarrow{t'_i} y_i \mid i \in I\}}{f(\overrightarrow{x}) \xrightarrow{t'} t''}$$

*and first, all* $x_i$ *and* $y_j$ *variables* $(0 \leq i < ar(f)$ *and* $j \in I)$ *and variables in* $t'$ *are pairwise distinct, second, if a component of* $t_i$ $(i \in I)$ *is a variable (i.e., does not have any function symbol) then it is not among* $x_i$*'s and* $y_j$*'s and third,*

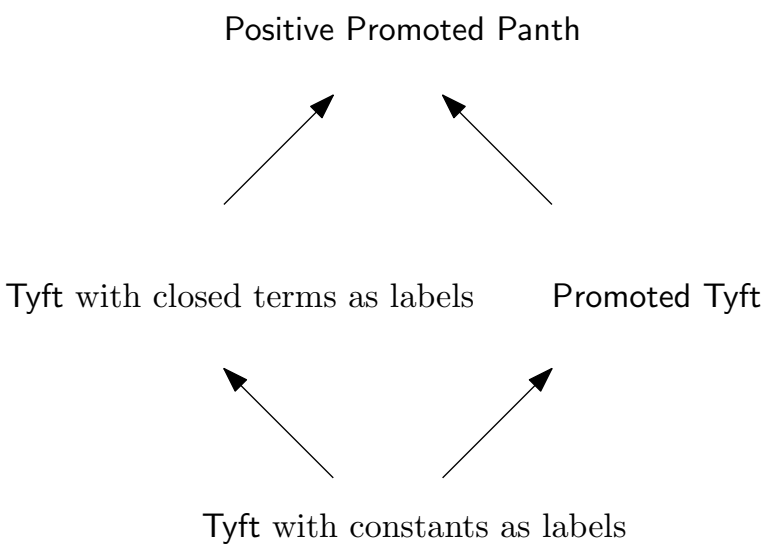



Fig. 1. Comparison of the expressiveness of rule formats.

(i) *if $t'$ contains a volatile $g \in \Sigma$ then $t'$ is of the form $g(\overrightarrow{z})$ where all $z_i$'s are distinct variables and for all $j \in I$, all $t_i$ containing a variable among $\overrightarrow{z}$ are of the form $g_i(\overrightarrow{t'_i})$ where $g_i$ is volatile,*

(ii) *if there is a volatile operator in the signature and if $t'$ is a variable $z$ then for all $i \in I$, $t_i$ containing $z$ are either $z$ itself or are of the form $g_i(\overrightarrow{t'_i})$ where $g'$ is volatile.*

It follows immediately from the above definition that any TSS in the tyft format is in the positive promoted PANTH format since a TSS in the tyft format contains no volatile operator. On the other extreme resides the promoted tyft format which is a subset of positive promoted PANTH in which all operators are considered volatile (regardless of whether or not they actually are volatile). Thus, we conclude that positive promoted PANTH is strictly more expressive than both tyft and promoted tyft since it includes TSS's of examples of Section 4.1 and has both formats as its (proper) subsets.

Figure 1 summarizes the result of our comparison. Each arrow shows strict inclusion of the sets of definable transition relations.

## 5 Conclusions

In this paper we studied issues related to the well-foundedness of premises and expressiveness for (the set of transition relation that can be specified by) TSS's in the promoted tyft format. We showed that well-foundedness cannot be dropped while preserving the congruence property for bisimilarity. Furthermore, we compared the expressiveness of the tyft, the promoted tyft, and the positive subset of the promoted PANTH formats and showed that while the tyft format with closed terms is incomparable to the promoted tyft format, the positive subset of the promoted PANTH format is strictly more expressive

than both.

Regarding well-foundedness, we are currently studying the congruence meta-theorem for the safe subset of the promoted tyft format. The techniques used in [4] are not directly applicable to this setting as the open terms on the labels (containing at least one function symbol) cannot be trivially resolved to variables. Regarding expressiveness, it is interesting to compare the safe promoted tyft format with the promoted tyft format. We do not yet know the answer but expect the two formats to be equally expressive.

## References

[1] K. L. Bernstein, A congruence theorem for structured operational semantics of higher-order languages, in: Proceedings of the 13th IEEE Symposium on Logic In Computer Science (LICS'98), IEEE Computer Society, Los Alamitos, CA, USA, 1998, pp. 153–164.

[2] M.R. Mousavi, M. J. Gabbay, M. A. Reniers, SOS for higher order processes, in: Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05), Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany, 2005, pp. 308–322.

[3] J. F. Groote, F. W. Vaandrager, Structured operational semantics and bisimulation as a congruence, Information and Computation (I&C) 100 (2) (1992) 202–260.

[4] W. J. Fokkink, R. J. van Glabbeek, Ntyft/ntyxt rules reduce to ntree rules, Information and Computation (I&C) 126 (1) (1996) 1–10.

[5] D. J. Howe, Proving congruence of bisimulation in functional programming languages, Information and Computation (I&C) 124 (1996) 103–112.

# Modular Structural Operational Semantics with Strategies ⋆

Christiano Braga [1] Alberto Verdejo [2]

*Facultad de Informática*
*Universidad Complutense de Madrid*

**Abstract**

Strategies are a powerful mechanism to control rule application in rule-based systems. For instance, different transition relations can be defined and then combined by means of strategies, giving rise to an effective tool to define the semantics of programming languages. We have endowed the Maude MSOS Tool (MMT), an executable environment for modular structural operational semantics, with the possibility of defining strategies over its transition rules, by combining MMT with the Maude strategy language interpreter prototype. The combination was possible due to Maude's reflective capabilities. One possible use of MMT with strategies is to execute Ordered SOS specifications. We show how a particular form of strategy can be defined to represent an OSOS order and therefore execute, for instance, SOS specifications with negative premises. In this context, we also discuss how two known techniques for the representation of negative premises in OSOS become simplified in our setting.

*Key words:* Modular SOS, Strategies, Ordered SOS, Negative Premises

## 1 Introduction

Strategies are a powerful mechanism for the specification of programming languages and systems. A strategy language describes how rules should be applied in a given rule-based specification by means of a combination of basic strategies. In Maude's strategy language [7], our language of choice, a basic strategy specifies that a rule, denoted by its label, can be applied possibly with a given substitution and using given strategies to solve its premises, if any.

⋆ Research supported by MCyT Spanish project MIDAS (TIC2003-0100) and Ramón y Cajal program.

[1] Email: 'cbraga@fdi.ucm.es' (On leave from Universidade Federal Fluminense, Brasil.)

[2] Email: 'alberto@sip.ucm.es'

Strategy combinators are tests, conditionals, decomposition (i.e. a strategy applied to subterms), and search. Recursive strategies can also be defined. Non-trivial examples where the Maude's strategy language has been used to implement structural operational semantics are the Eden language, that has several transition relations which can be specified and combined by means of strategies [5], and the ambient calculus, where strategies [14] are used to control communication, replication and termination.

We have endowed Modular SOS (MSOS) [10] specifications with strategies, by putting together the Maude MSOS Tool (MMT) [2], an executable environment for MSOS, with Maude's strategy language (MSL) [7]. The combined tool, named MMT+MSL, is implemented as a conservative extension of Maude's extensible module algebra implemented in Full Maude [4]. To illustrate the usefulness of our proposal, we show how Ordered SOS (OSOS) [15] specifications can be directly represented in MMT+MSL, where the transition rules are the same and the order is represented as a strategy. Then, using this representation, negative premises become executable in MMT+MSL by the application of the techniques given in [15], and yet, simplified. As a concrete example, we extend the modular SOS specification of CCS with priorities.

This paper is organized as follows. Section 2 overviews Maude's strategy language, exemplifies the syntax for specifications accepted by MMT+MSL, using CCS as example, and the implementation MMT+MSL in Full Maude. Section 3 explains how Ordered SOS specifications can be represented as specifications in MMT+MSL. Section 4 briefly recalls how negative premises can be represented in OSOS. Section 5 extends the CCS specification in Section 2 with a priority operator. Section 6 concludes the paper with our final remarks.

## 2 MMT+MSL

### *2.1 Maude's Strategy Language*

Rewrite rules in rewriting logic need be neither confluent nor terminating. This theoretical generality requires some control when the specifications become executable, because it must be ensured that the rewriting process does not go in undesired directions. Maude's strategy language can be used to control how rules are applied to rewrite a term [7]. The simplest strategies are the constants '`idle`', which always succeeds by doing nothing, and '`fail`', which always fails. The basic strategies consist of the application of a rule (identified by the corresponding rule label) to a given term, and with the possibility of providing a substitution for the variables in the rule. In this case a rule is applied *anywhere* in the term where it matches satisfying its condition. When the rule being applied is a conditional rule with rewrites in the conditions, the strategy language allows to control how the rewrite conditions are solved by means of search expressions. An operation '`top`' to restrict the application of a rule just to the *top* of the term is also provided. Basic strategies are then

combined so that strategies are applied to execution paths. Some strategy combinators are the typical regular expression constructions: concatenation (‘`;`’), union (‘`|`’), and iteration (‘`*`’ for 0 or more iterations, ‘`+`’ for 1 or more, and ‘`!`’ for a ‘repeat until the end’ iteration). Another strategy combinator is a typical ‘if-then-else’, but generalized so that the first argument is also a strategy. By using this combinator, we can define many other useful strategy combinators as derived operations: for example a binary ‘`orelse`’ combinator that applies the second argument strategy only if the first fails, and a unary ‘`not`’ combinator that fails when its argument is successful and vice versa. The language also provides a ‘`matchrew`’ combinator that allows a term to be split in subterms, and specifies how these subterms have to be rewritten. An extended ‘`matchrew`’, ‘`xmatchrew`’, is also provided where rewriting modulo axioms associativity, commutativity, identity and idempotency is considered, when declared. Recursion is also possible by giving a name to a strategy expression and using this name in the strategy expression itself or in other related strategies.

Using the Maude metalevel, we have implemented a prototype of the strategy language as an extension of Full Maude [7]. Currently the language is being integrated in the Maude system.

### *2.2 CCS in MMT+MSL*

Modular SOS is a variant of SOS that allows for specifications to be made modular by structuring the labels in the transition rules as extensible records. Semantic rules for a given constructor use certain indices from the record structure, so that newly added rules could range over (existing or) new indices, thus allowing that existing rules are not changed when new semantic entities are required. Therefore, semantic rules may be declared *once and for all*. For instance, rules for a functional fragment may access an environment from the label structure while rules for an imperative fragment may access the memory component.

MMT [1] is an executable environment for MSOS and was implemented as a formal tool in the precise sense presented in [3], that is, as a realization of a semantics preserving mapping between Modular SOS and rewriting logic. The modular SOS definition formalism is the specification supported by MMT. It allows MSOS specifications to be written in a quite succinct syntax that includes: support for grammar specification in BNF like syntax, implicit module inclusion, “type declaration” as alias for instantiated parameterized built-in types, automatic derived set and list declarations for each explicitly declared set in the BNF or aliasing sections, automatic variable declarations by appending “primes” and numbers on the set names, and explicit label structure declaration.

Let us discuss now how CCS can be specified and executed in MSDF. We also present a strategy that solves the rules premises in depth-first search.

Concrete labels and process identifiers are declared to test the execution of our specification. No runs are shown in the paper but the tool and this example can be downloaded from http://maude-msos-tool.sf.net/mmt+msl/.

We follow the constructive approach for semantic descriptions proposed by Mosses in [11] and thus present each construct as a separate module in MSDF.

The module 'LABEL' declares a set for action labels. The module 'ACTION' declares the set 'Action' that includes labels and the (unobservable) 'tau' action.

```
(msos LABEL is
  Label .
  Label ::= ~ Label | a | b | c .
sosm)
```

```
(msos ACTION is
  Action .
  Action ::= Label | tau .
sosm)
```

The module 'PROCESS' declares the set of processes ('Process') and the idle process ('0').

```
(msos PROCESS is
  Process .
  Process ::= 0 .
sosm)
```

The MSOS label structure used in the modules below has an index 'trace'' representing the process trace. The quote in 'trace'' has a meaning: in MSOS terminology it is a write-only index, that is, it can only be updated.

Transition rules in MSDF represent quite directly standard mathematical notation for transition rules. A few explanations may clarify, however, the notation for label patterns. Labels may have ellipsis (...) or a dash (-) to represent all the indices in a label not explicitly mentioned. When ellipsis are used it means that the part of the label it refers to may be changed in a transition. The dash is used otherwise. When they occur more than once in the same rule, they refer to the same subset of the indices. Metavariables, such as X1 and X2, may also be used to refer to a subset of the indices of a label and are used to distinguish between two sets of indices in the same rule.

The module 'PREFIX' declares an action prefix (';') that adds an action to the trace. Note that the set 'Action*', for a possibly empty set of actions, has not been declared explicitly. It was automatically derived by the declaration of the set 'Action' in module 'ACTION', which was automatically imported by 'PREFIX'.

```
(msos PREFIX is
  Process ::= Action ; Process [prec 20] .
  Label = {trace' : Action*, ...} .

  [prefix] (Action ; Process) : Process -{trace' = Action,-}-> Process .
sosm)
```

Summation ('+') means simply to choose one of the processes to evolve. Note that only one rule is needed since the operator is declared as commutative, with keyword 'comm' in the BNF declaration.

```
(msos SUMMATION is
  Process ::= Process + Process [assoc comm prec 30] .
  Label = {trace' : Action*, ...} .

           Process1 -{...}-> Process1'
  [sum] -- ------------------------------------------------
        (Process1 + Process2) : Process -{...}-> Process1' .
sosm)
```

Parallelism ('`||`') allows one process to evolve or both if they synchronize, that is, one performs '`Action`' and the other '`~Action`'. The CCS semantics does not specify how synchronization behaves in the presence of side-effects. In our semantics no side-effects may be produced while synchronizing. (This is the semantics for synchronization in Reppy's $\lambda_{cv}$, for instance, whose MSOS semantics is given in [9].)

```
(msos PARALLELISM is see ACTION .
  Process ::= Process || Process [assoc comm prec 25] .
  Label = {trace' : Action*, ...} .

           Process1 -{...}-> Process1'
  [par1] -- ------------------------------------------------------------
         (Process1 || Process2) : Process -{...}-> Process1' || Process2 .

           Process1 -{trace' = Action, -}-> Process1' ,
           Process2 -{trace' = ~ Action, -}-> Process2'
  [par2] -- ------------------------------------------------------------
         (Process1 || Process2) : Process -{trace' = tau, -}->
                                                 Process1' || Process2' .
sosm)
```

Relabeling ('`rel`') substitutes a performed action label by another one.

```
(msos RELABELLING is see ACTION .
  Process ::= rel (Process, Label, Label) [prec 20] .
  Label = {trace' : Action*, ...} .

           Process1 -{trace' = Action1, ...}-> Process1'
  [rel1] -- ------------------------------------------------------------
         (rel (Process1, Action2, Action1)) : Process
                                -{trace' = Action2, ...}-> Process1' .

           Process1 -{trace' = ~ Action1, ...}-> Process1'
  [rel2] -- ------------------------------------------------------------
         (rel (Process1, Action2, Action1)) : Process
                               -{trace' = ~ Action2, ...}-> Process1' .

           Process1 -{trace' = Action3, ...}-> Process1' ,
           Action3 =/= Action1,
           Action3 =/= ~ Action1
  [rel3] -- ------------------------------------------------------------
         (rel (Process1, Action2, Action1)) : Process
                                -{trace' = Action3, ...}-> Process1' .
sosm)
```

Finally, restriction ('\') of an action means that a process is allowed to evolve if it does not signal the given action or its negation.

```
(msos RESTRICTION is see ACTION .
  Process ::= Process \ Label [prec 25] .
  Label = {trace' : Action*, ...} .

           Process1 -{trace' = Label2, ...}-> Process1' ,
           Label2 =/= Label1,
           Label2 =/= ~ Label1
  [res] -- ------------------------------------------------------------
        (Process1 \ Label1) : Process
                          -{trace' = Label2, ...}-> Process1' \ Label1 .
sosm)
```

MSDF is implemented in MMT as a conservative extension of Full Maude. Therefore functional modules (for equational specifications) and system modules (for equational and rule-based specifications) in Maude may be used together with MSDF specifications. Double-negation of labels are specified as an equation in the functional module 'LABEL-CONGRUENCE' which is then combined with the above MSDF modules in the 'CCS' system module.

```
(fmod LABEL-CONGRUENCE is
   inc LABEL .
   eq ~ ~ Label:Label = Label:Label .
 endfm)
```

```
(mod CCS is
  inc PROCESS . inc PREFIX . inc SUMMATION .
  inc PARALLELISM . inc RELABELLING .
  inc RESTRICTION . inc LABEL-CONGRUENCE .
 endm)
```

Before we explain the details of the strategy module, a word is needed on how to represent Modular SOS computations in Maude. Maude implements the rewriting logic calculus which has four inference rules given by reflexivity (a term can be rewritten to itself), transitivity (if $t$ rewrites to $t'$ and $t'$ to $t''$, then $t$ rewrites to $t''$), congruence (a rule can be applied to the subterms of $t$), and substitution (a rule can be applied to a kind preserving substitution). SOS does not have such a calculus. The present authors, with others, have proposed several techniques (e.g. [16,8]) to represent both modular and plain SOS computations in rewriting logic and have implemented them in Maude. Using a strategy, however, these techniques are not necessary since one has full control of the rule application. Reflexivity and transitivity are controlled by basic strategies, that is, if a basic strategy is applied, it represents one (rewriting) step. Congruence, however, needs to be controlled, that is, the application of a basic strategy should be done at the *top* operator and not on its subterms. That is why the 'top' strategy is applied. The substitution inference rule is desired and therefore needs not to be controlled.

Instead of using the 'top' strategy, we could have also used the technique implemented in MMT [8] to control Maude's default rewriting strategy, which is essentially a rewrite rule (labeled 'step') with extra configuration constructors that impose a one-step rewrite for each rule application. It simplifies the strategy but adds extra declarations related to the 'step' rule to the gener-

ated Maude module. Thus, the choice for the '`top`' operator produces cleaner Maude modules.

When applying a rule with premisses, the strategy should specify which is the strategy applied to solve each premise. In order to make the strategy extensible, we use a "abstract" strategy that will be instantiated later on. [3]

```
(stratdef PREM-STRAT is
  sop prem-strat .
 endsd)
```

Another implementation detail is that here we make explicit that the strategy '`prem-strat`' is applied in depth-first search to the premises of the transition rules. Another alternative could be to use breadth-first search if infinite recursive processes were allowed by using contexts.

Thus, the complete strategy is given by the union strategy '`|`' of the basic strategies for each operator together with the remarks for the '`top`' strategy and the evaluation of premises described above.

```
(stratdef CCS-STRAT is
  including CCS . including PREM-STRAT .
  sop ccs-strat .
  seq ccs-strat = top(prefix)
           | top(sum{dfs(prem-strat)})
           | top(par1{dfs(prem-strat)})
           | top(par2{dfs(prem-strat) dfs(prem-strat)})
           | top(rel1{dfs(prem-strat)})
           | top(rel2{dfs(prem-strat)})
           | top(rel3{dfs(prem-strat) dfs(prem-strat)})
           | top(res{dfs(prem-strat)}) .
endsd)
```

Note that this strategy can be automatically generated by inspecting the semantics rules. It reflects the MSOS derivation mechanism, but it is not CCS dependent. Moreover, the strategy `ccs-strat` can only be used after concretizing the strategy `prem-strat` as the following module does.

```
(stratdef CCS-STRAT+ is
  including CCS-STRAT .
  seq prem-strat = ccs-strat .
endsd)
```

This mechanism, that allows the modular definition of the strategies, will be further exemplified below when CCS will be extended with a priority operator in Section 5.

### *2.3 The Implementation of MMT+MSL*

The current version of MMT+MSL relies on the prototypes for MMT and MSL implemented in Full Maude. As we mentioned before, Full Maude implements

[3] What we really need is a parameterized strategy module, but the extension of the Maude strategy language with parametric modules is currently under study.

an extensible module algebra for Maude. It provides a basic infrastructure, that is, a set of meta-functions, to extend Maude, that relies on Maude's meta-programming interface. (For instance, '`metaParse`' produces a term out of a given list of identifiers and a grammar.)

The Maude predefined '`LOOP-MODE`' module defines a read-eval-print loop that should be extended in order to define a command-line interface for a Maude extension. It defines a triple containing the input (of sort '`QidList`'), the current state of the system (of sort '`State`'), and the output (of sort '`QidList`'), given by the infix operator '`[_,_,_] : QidList State QidList -> System`'. The descent functions above should then manipulate these values. This is what Full Maude does, as described below.

In the reminder, we first comment on the general technique to extend Maude, and then move to Full Maude. The following steps should be done: to define a module $M$ representing the syntax of the language that one wants to represent in Maude; to define a meta-function that given a meta-term in the meta-representation of $M$, produces a meta-term in the meta-representation of a Maude module; to define an interface that encapsulates how commands in the language captured by $M$ are translated into commands over the Maude representation of $M$; and how the "answer" given by the Maude system is translated back into the language of $M$.

Full Maude provides an infrastructure to implement all these steps. There is a parsing infrastructure to handle Maude-based modules; a transformation infrastructure that given a structured Maude module, that is, a Maude module with inclusions, flattens it into a single Maude module; a database, that is, a term that holds all the modules loaded in Full Maude, together with information necessary to execute Full Maude's commands (the database structure may be extended to "cache" information that may be necessary for computing with (the representation of) terms in $M$); and finally a pretty-printing infrastructure. This infrastructure is already used by Full Maude to specify parameterized modules, and object-oriented modules for instance.

MMT and MSL were implemented as Full Maude extensions individually. (Concrete details on how both tools have been implemented at the metalevel can be found in [2,7].) The combination was straightforward: we wrote a few modules that joined each of these parts, that is, parsing, transformation, database handling, and pretty printing. The module '`MMT+MSL-SIGN`' extends the module '`STRAT-GRAMMAR`' (that itself extends Full Maude's grammar with the one for strategies) with the grammar for MSDF syntax defined in module '`MSDF-SIGNATURE`'.

```
fmod MMT+MSL-SIGN is
  including META-STRAT-SIGN .
  op MMT+MSL-GRAMMAR : -> FModule .
  eq MMT+MSL-GRAMMAR = addImports((including 'MSDF-SIGNATURE .), STRAT-GRAMMAR) .
endfm
```

The module '`MMT+MSL`' puts the Maude modules from both tools together.

It replaces Full Maude's module for handling input and output since there can not be non-determinism between Full Maude's rules and MMT+MSL. First it includes the extended grammar, then the database handling modules for MSL and MMT, the predefined units for MMT and finally the loop mode module.

```
mod MMT+MSL is
  pr MMT+MSL-SIGN .           pr STRAT-DATABASE-HANDLING .
  pr MMT-DATABASE-HANDLING .  pr PREDEF-UNITS .
  inc LOOP-MODE .
```

Three rules handle the read-eval-print loop for MMT+MSL. The rule labeled '`init`' below simply initializes Full Maude's database with its default values and adds a "banner" to the MMT+MSL. Full Maude's database is the state of the system declared by the '`LOOP-MODE`' module. It uses Maude object-oriented notation. The database structure was extended both by MSL and the MMT. The attributes '`state`', '`stratDefs`', '`results`', and '`repeat`' are used by MSL to, respectively, represent the search tree (either a stack, representing backtrack points, for depth-first search or a queue, with unsolved terms, for breadth-first search), a meta-module representing the strategy definitions, a set of terms representing the solutions found so far, and a flag for the option of showing or not repeated results. The attribute '`step-flag`' is declared by MMT and holds the option to use MMT's built in technique to handle MSOS computations (the '`step`' rule) or not.

```
  rl [init] : init
   => [nil, < o : STRATDB | db : initial-db, input : nilTermList, output : nil,
              default : 'CONVERSION, state : emptyP, step-flag : false,
              stratDefs : none, results : emptyTermSet, repeat : false >,
          ('\n '\t '\s '\s '\s '\s '\s '\s '\s
           'MMT 'and 'Strategy 'Full 'Maude '2.1.1 'Combined '\s  '\n)] .
```

Rule '`in`' allows for both modules to be entered in MMT+MSL by invoking '`metaParse`' with the combined grammar in module '`MMT+MSL-GRAMMAR`'. There is another '`in`' rule that handles syntax errors in the input.

```
  crl [in] :
    [QIL, < O : X@Database | input : nilTermList, output : nil, Atts >, QIL']
    => [nil, < O : X@Database |
               input : getTerm(metaParse(MMT+MSL-GRAMMAR, QIL, 'Input)),
               output : nil, Atts >, QIL']
    if QIL =/= nil /\  metaParse(MMT+MSL-GRAMMAR, QIL, 'Input) : ResultPair .
```

Rule '`out`' simply prints to the screen what was produced as output by Full Maude.

```
  crl [out] :
    [QIL, < O : X@Database | output : QIL', Atts >,  QIL'']
    => [QIL, < O : X@Database | output : nil, Atts >, (QIL'' QIL')]
    if QIL' =/= nil .
endm
```

Finally, the MMT+MSL tool can be used after loading into Maude the modules implementing MMT and MSL and the two modules described above.

## 3 Representing Ordered SOS with Strategies

In this section we show how a Maude strategy can be defined to execute an ordered SOS specification as defined in [15].

A set of rules with an ordering (any binary relation) is called *ordered SOS* (OSOS) if it contains positive GSOS rules only (that is, no rule has negative premises). In [15] it is shown that GSOS and OSOS have the same expressive power. A GSOS rule is an inference rule in the following form

$$\frac{\left\{ X_i \xrightarrow{\alpha_{ij}} Y_{ij} \right\}_{i\in I, j\in J_i} \quad \left\{ X_k \overset{\beta_{kl}}{\not\rightarrow} \right\}_{k\in K, l\in L_k}}{f(X_1,\ldots,X_n) \xrightarrow{\gamma} C[\mathbf{X},\mathbf{Y}]}$$

where $I$ and $K$ are subsets of $\{1,\ldots,n\}$ and all $J_i$ and $L_k$ are finite subsets of $\mathbb{N}$; $\mathbf{X}$ is the sequence $X_1,\ldots,X_n$ and $\mathbf{Y}$ is the sequence of all $Y_{ij}$; and $C$ is a context.

For example, the following rules define the behavior of an hypothetical operator $f$, for constants $a$ and $b$:

$$\frac{X \xrightarrow{a} Y}{f(X) \xrightarrow{a} f(Y)}\ r_1 \qquad \frac{X \xrightarrow{b} Y}{f(X) \xrightarrow{b} g(Y)}\ r_2 \qquad \{r_1 < r_2\}$$

where the relation $r_1 < r_2$ between the rules specifies that the first rule ($r_1$) is only applied when the second rule ($r_2$) cannot be applied. That is, the binary relation on rules defines the *order of their application* when deriving transitions. So, a rule $r$ can be used to derive a transition if all its premises are valid and no rule *higher* than $r$ is applicable (it contains a premise which is not valid).

A Maude strategy can be used to take into account this order on rules. First, inference rules are represented as SOS rules in MMT but their application will be controlled by a strategy. The strategy makes use of the '`top`' combinator to restrict the application of the given strategy to the outermost term. The '`not`' combinator checks if the higher rules cannot be applied.

For the previous example with $r_1 < r_2$, the part of the strategy that tries to apply $r_2$ is simply `top(`$r_2$`{`$s$`})`, where $s$ is the abstract strategy to be used to solve the premise. The part of the strategy regarding the application of $r_1$ is a bit more complex since it has rules higher in the rule ordering. It is '`not(top(`$r_2$`{`$s$`})) ; top(`$r_1$`{`$s$`})`', which means that before applying $r_1$ we have to know that $r_2$ cannot be applied. The strategy '`not(top(`$r_2$`{`$s$`}))`' succeeds if `top(`$r_2$`{`$s$`})` fails.

For an OSOS specification $(\Sigma, A, R, <)$ the following algorithm builds the strategy identified by $s$ that controls the way rules in $R$ should be applied. It uses a function $rules(f)$ to obtain the rules in $R$ that define the behavior of the operator $f$ and a function $higher(r)$ that returns all the rules $r' \in R$ such that $r < r'$.

strategy := '`idle`'
for each operator $f \in \Sigma$ do
  for each rule $r_f \in rules(f)$ do
    if $higher(r_f) = \emptyset$ then
      append '`| top(`$r_f$`{`$s$`})`' to the strategy
    else
      append '`| not(`$union(higher(r_f))$ `) ; top(`$r_f${s}$`)`' to the strategy

where $union(r_1, \ldots, r_n) =$ `top(`$r_1${s}$`) | ... | top(`$r_n${s}$`)`.

The strategy '`not(top(`$r_1${s}$`) | ... | top(`$r_n${s}$`))`' succeeds when none of the rules $r_1, \ldots, r_n$ can be applied. Note that if a given rule $r$ has $m > 1$ premises then the strategy $s$ should be repeated $m$ times within the curly brackets. If $m = 0$ then $r$ is not parameterized.

In [15] a transition relation $\rightarrow$, that takes into account the ordering on rules, is associated to a process language $(\Sigma, A, R, <)$. Formally, $\rightarrow = \bigcup_{l<\omega} \rightarrow^l$, where the transitions in $\rightarrow^l$ are defined as follows

$$p \xrightarrow{\alpha} p' \in \rightarrow^l \text{ if } d(p) = l \text{ and } \exists r \in R, \rho.\Big(\rho(con(r)) = p \xrightarrow{\alpha} p' \text{ and } \rho(pre(r)) \subset \bigcup_{k<l} \rightarrow^k \text{ and } \forall r' \in higher(r).\rho(pre(r')) \not\subset \bigcup_{k<l} \rightarrow^k \Big).$$

**Theorem 3.1** *The transition relation induced by an OSOS specification is preserved by the associated SOS specification with the strategy built by the above algorithm.*

**Proof sketch.** By induction on the depth of the process term. The base case is when the process is a constant, and therefore the rule $r$ does not have any premise and $higher(r) = \emptyset$. The strategy produced by the algorithm has $r$ as one of its alternatives. For the inductive case, since $\forall r' \in higher(r).\rho(pre(r')) \not\subset \bigcup_{k<l} \rightarrow^k$ holds, then by inductive hypothesis the strategy `not(`$union(higher(r))$`)` succeeds since the application of each rule $r'$ fails; and also, $\rho(pre(r)) \subset \bigcup_{k<l} \rightarrow^k$ holds, thus by inductive hypothesis the strategy that is applied to the premisses of $r$ succeeds, which makes the application of strategy $r${s}`` successful. □

## 4 Representing Negative Premises

In this section we first recall how a GSOS specification with negative premises can be represented in OSOS and then how a strategy can be used for that matter. We adapt material from [15] while recalling how negative premises are represented in OSOS.

For OSOS specifications with no constraints whatsoever regarding the rule ordering (besides being simply a binary relation among rules), given a rule with a negative premise, a new rule is generated above the given rule in the

rule order. Its single premise is a positive version of the negative premise in the given rule. As for its conclusion, it has the same left-hand side of the conclusion of the given rule, but with process **0** on the right-hand side. Moreover, the generated rule should never be enabled for a configuration where its premise holds, hence, it should be above itself in the rule order.

Let us consider the specification for an hypothetical operator $f$ given by the following rule:

$$\frac{X \xrightarrow{b} Y \quad X \overset{a}{\not\to}}{f(X) \xrightarrow{b} t'}\ r_1$$

This specification can be written in OSOS simply by removing the negative premise from rule $r_1$, declaring the rule $r_2$ below, and an order where $r_2$ is above $r_1$. The specification then becomes as follows, where $Y$ is a new variable in $r_2$.

$$\frac{X \xrightarrow{b} Y}{f(X) \xrightarrow{b} t'}\ r_1 \qquad \frac{X \xrightarrow{a} Y}{f(X) \xrightarrow{a} t'}\ r_2 \qquad \{r_1 < r_2,\ r_2 < r_2\}$$

Our SOS specification with a strategy is then given by rules $r_1$ and $r_2$, as above, together with the strategy '$s = \texttt{not}(r_2\{s\})\ ;\ r_1\{s\}\ |\ \texttt{not}(r_2\{s\})\ ;\ r_2\{s\}$'. (The abstract strategy technique is not used here for simplicity.)

Clearly, the strategy '$\texttt{not}(r_2\{s\})\ ;\ r_2\{s\}$' is not necessary (since it always fails) and the strategy could be simplified. Also, note that rule $r_2$ is really *never* applied. In the strategy '$\texttt{not}(r_2\{s\})\ ;\ r_1\{s\}$' the strategy '$\texttt{not}$' only checks if the premises hold. Another remark is that a rule for the operator $f$ with the premise of $r_2$ could already exist in the original GSOS specification. In this case the specification is called *natural* in [15]. Thus, from natural specifications is not necessary to generate a new rule and the strategy could simply take the existing rule into account.

OSOS specifications can be *partial*, meaning that its order is irreflexive and transitive. (Partial OSOS specifications are also equivalent to GSOS specifications according to [15].) Since the order has to be partial, the technique of having a rule above itself can not be used. The technique to represent the negative premise in partial OSOS relies on an extended action set with an *error* action and a rule that restricts process evolution to processes that do not signal *error*. A rule is also generated in the form of the one produced by the technique for non-partial OSOS, which is also above the given rule in the rule order, but with the *error* action in the conclusion. Also, the initial configuration should be augmented with the restriction to *error*.

The specification for $r_1$ in partial OSOS is given by the following three rules:

$$\frac{X \xrightarrow{b} Y}{f(X) \xrightarrow{b} t'}\ r_3 \qquad \frac{X \xrightarrow{a} Y}{f(X) \overset{error}{\to} \mathbf{0}}\ r_4 \qquad \{r_3 < r_4\}$$

$$\frac{X \xrightarrow{\alpha} Y}{X\backslash error \xrightarrow{\alpha} Y\backslash error}\, r_5 \quad \alpha \neq error$$

where the initial configuration with process $f(p)$ should be augmented with the restriction to action *error*, as in $f(p)\backslash error$.

In this case the strategy would be '$s$ = `not(`$r_4$`{`$s$`}) ; `$r_3$`{`$s$`} | `$r_5$`{`$s$`}`'. Again, a simplification is possible, due to the same reason as for the non-partial case. Since the strategy '`not`' does not apply a rule, only checks for its premises, the conclusion of $r_4$ is irrelevant. Therefore there is neither a need to extend the action set with the *error* action nor to add $r_5$ to the rule set. With this simplification the resulting strategy is '$s$ = `not(`$r_4$`{`$s$`}) ; `$r_3$`{`$s$`}`'.

Both translations (including the generation of new rules and the corresponding order) can be done automatically by inspecting the GSOS rules. Then the strategy can be generated as explained in Section 3. (The implementation of this transformation, however, is part of future work.)

## 5 CCS with Priority

### *5.1 A Priority Operator with Strategies*

An example of the usage of negative premises is a priority operator $\theta$ [12], which given a process $P$ builds a new process that performs action $\alpha$ of $P$ if $P$ cannot perform any action with a priority higher than $\alpha$. This operator is specified by the following rule scheme $r_\theta$.

$$\frac{X \xrightarrow{\alpha} X' \quad \forall_{\beta>\alpha}\ X \overset{\beta}{\not\rightarrow}}{\theta(X) \xrightarrow{\alpha} \theta(X')}\, r_\theta$$

Given a finite set of actions, the above scheme can be represented by many rules like $r_\theta$ but without the negative premise and with an ordering among them. An example strategy for $r_\theta$ is '$s$ = `not(`$r_{\theta_c}$`{`$s$`} | `$r_{\theta_b}$`{`$s$`}) ; `$r_{\theta_a}$`{`$s$`} | not(`$r_{\theta_c}$`{`$s$`}) ; `$r_{\theta_b}$`{`$s$`} | `$r_{\theta_c}$`{`$s$`}`', given a set of action labels $\{a, b, c\}$ with the ordering $\{a < b, a < c, b < c\}$, and the rules for the priority operator labeled $r_{\theta_a}$, $r_{\theta_b}$, and $r_{\theta_c}$. (Again the abstract strategy technique is not applied for simplicity.)

However, this specification can be further simplified. Strategies may be applied with a particular *substitution*. Thus, instead of having three rules, in this example, we may specify a single rule $r_\theta$ with an action variable that may become bound to the three different label actions, thus giving rise to the strategy '$s$ = `not(`$r_\theta$`[A` $\leftarrow$ $c$`]{`$s$`} | `$r_\theta$`[A` $\leftarrow$ $b$`]{`$s$`}) ; `$r_\theta$`[A` $\leftarrow$ $a$`]{`$s$`} | not(`$r_\theta$`[A` $\leftarrow$ $c$`]{`$s$`}) ; `$r_\theta$`[A` $\leftarrow$ $b$`]{`$s$`} | `$r_\theta$`[A` $\leftarrow$ $c$`]{`$s$`}`', where '`A`' is an action variable.

For arbitrary large (but finite) set of actions, the strategy could be parameterized by a list of action labels representing the action labels above a

given one. If we consider the following function *forall* below that produces a strategy out of a list of action labels, the strategy for an action label $a$ with the function application $higher(a)$ returning a list of action labels, would be given by the expression '`not(`$forall(higher(a))$ `)  ;  `$r_\theta$'.

$$forall_{r_\theta}(l, ls) = r_\theta\,\texttt{[A} \leftarrow l\texttt{]} \;\; \texttt{|} \;\; forall_{r_\theta}(ls)$$
$$forall_{r_\theta}(nil) = idle$$

### 5.2 CCS with the Priority Operator in MMT+MSL

The specification of CCS in Section 2.2 can be very easily extended, given the representation of priorities as strategies in Section 5.1. First the syntax of processes must be extended with the priority operator and the transition rule set must be extended with a new transition rule for priorities as $r_\theta$ in Section 5.1 but without the negative premises as we explained before. The Maude module '`CCS-PRI`', that includes the '`CCS`' module above and '`PROCESS-WITH-PRIORITY`' is also defined.

```
(msos PROCESS-WITH-PRIORITY is
  Process ::= theta (Process) [prec 20] .
  Label = {trace' : Action*, ...} .

             Process -{trace' = Action, ...}-> Process'
  [theta] -- ------------------------------------------------------------------
             theta (Process) : Process -{trace' = Action, ...}-> theta (Process') .
sosm)
```

The strategy has to be extended with the new strategies to represent the negative premises as explained in Section 5.1.

```
(stratdef PRI-STRAT is
  including PREM-STRAT .
  sop pri-strat .
  seq pri-strat = not(top(theta[Action <- c]{dfs(prem-strat)}) |
                      top(theta[Action <- b]{dfs(prem-strat)}))
                  ; top(theta[Action <- a]{dfs(prem-strat)})
                | not(top(theta[Action <- c]{dfs(prem-strat)}))
                  ; top(theta[Action <- b]{dfs(prem-strat)})
                | top(theta[Action <- c]{dfs(prem-strat)})) .
endsd)
```

The module '`CCS-PRI-STRAT`', that replaces '`CCS-STRAT+`', combines the strategy for basic CCS with the strategy for the priority operator, and establishes that the premises should be rewritten using the new whole strategy is the following one:

```
(stratdef CCS-PRI-STRAT is
  including CCS-STRAT .  including PRI-STRAT .
  sop ccs-pri .
  seq ccs-pri = ccs-strat | pri-strat .
  seq prem-strat = ccs-pri .
endsd)
```

## 6 Final Remarks

In [13] the authors present a prototype for GSOS specifications in Maude using the meta-level. Our approach represents OSOS, which is equivalent to GSOS [15], using the object level. Of course, it is still necessary to automate the translation from negative premises to orders and then to strategies. Moreover, to represent OSOS (and therefore GSOS) is *one possible* application of strategies. Maude (with strategies) could be used directly to represent any application with strategies, including OSOS. However, if one wants to make its specifications modular, the rewrite theories would have to be extended to cope with the modularity requirements.

The current version of the prototype does not support strategy module inclusion, even though there is notation (and semantics) for them in [7]. All the strategy definitions have to be declared in a single module. Part of our future work is to fully support the strategy language. Besides the automation of the translation of negative premises to strategies, a case study that we would like to approach in a near future is the implementation of E-LOTOS semantics [6], where negative premises are used in order to guarantee that urgent actions occur before time elapses.

**Acknowledgments**

We would like to thank Fabricio Chalub for his support on the implementation of MMT+MSL, the anonymous referees for their insightful comments and Peter Mosses for his careful review.

## References

[1] Chalub, F., "An Implementation of Modular Structural Operational Semantics in Maude," Master's thesis, Universidade Federal Fluminense (2005), http://www.ic.uff.br/~frosario/dissertation.pdf.

[2] Chalub, F. and C. Braga, *Maude MSOS tool*, in: G. Denker and C. Talcott, editors, *Proceedings Sixth International Workshop on Rewriting Logic and its Applications, WRLA 2006*, Electronic Notes in Theoretical Computer Science (2006), to appear.

[3] Clavel, M., F. Durán, S. Eker, J. Meseguer and M.-O. Stehr, *Maude as a formal meta-tool*, in: J. Wing, J. Woodcock and J. Davies, editors, *FM'99 — Formal Methods, Proc. World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 1999, Volume II*, Lecture Notes in Computer Science **1709** (1999), pp. 1684–1703.

[4] Durán, F., "A Reflective Module Algebra with Applications to the Maude Language," Ph.D. thesis, Universidad de Málaga (1999).

[5] Hidalgo-Herrero, M., A. Verdejo and Y. Ortega-Mallén, *Looking for Eden through Maude and its strategies*, in: F. López-Fraguas, editor, *V Jornadas sobre Programación y Lenguajes, PROLE 2005* (2005), pp. 13–23.

[6] ISO/IEC, *Information technology — Enhancements to LOTOS (E-LOTOS)*, International Standard ISO/IEC FDIS 15437 (2001).

[7] Martí-Oliet, N., J. Meseguer and A. Verdejo, *Towards a strategy language for Maude*, in: N. Martí-Oliet, editor, *Proceedings Fifth International Workshop on Rewriting Logic and its Applications, WRLA 2004, Barcelona, Spain, March 27 – April 4, 2004*, Electronic Notes in Theoretical Computer Science **117** (2005), pp. 417–441.

[8] Meseguer, J. and C. Braga, *Modular rewriting semantics of programming languages*, in: C. Rattray, S. Maharaj and C. Shankland, editors, *Algebraic Methodology and Software Technology: 10th International Conference, AMAST 2004, Stirling, Scotland, UK, July 2004, Proceedings*, Lecture Notes in Computer Science **3116** (2004), pp. 364–378.

[9] Mosses, P. D., *A modular SOS for ML concurrency primitives*, Technical Report BRICS-RS-99-57, Department of Computer Science, University of Aarhus (1999).

[10] Mosses, P. D., *Modular structural operational semantics*, Journal of Logic and Algebraic Programming **60-61** (2004), pp. 195–228.

[11] Mosses, P. D., *A constructive approach to language definition*, Journal of Universal Computer Science **11(7)** (2005), pp. 1117–1134.

[12] Mousavi, M., "Structuring Structural Operational Semantics," Ph.D. thesis, Technische Universiteit Eindhoven (2005).

[13] Mousavi, M. and M. A. Reniers, *Prototyping SOS meta-theory in Maude*, in: P. D. Mosses and I. Ulidowski, editors, *Proceedings of the Second Workshop on Structural Operational Semantics (SOS 2005)*, Electronic Notes in Theoretical Computer Science **156(1)** (2006), pp. 135–150.

[14] Rosa-Velardo, F., C. Segura and A. Verdejo, *Typed mobile ambients in Maude*, in: H. Cirstea and N. Martí-Oliet, editors, *Proceedings of the 6th International Workshop on Rule-Based Programming (RULE 2005)*, Electronic Notes in Theoretical Computer Science **147** (2006), pp. 135–161.

[15] Ulidowski, I. and I. Phillips, *Ordered SOS process languages for branching and eager bisimulations*, Information and Computation **178** (2002), pp. 180–213.

[16] Verdejo, A. and N. Martí-Oliet, *Executable structural operational semantics in Maude*, Journal of Logic and Algebraic Programming **67** (2006), pp. 226–293.

# An Eclipse-based Integrated Environment for Developing Executable Structural Operational Semantics Specifications

Adrian Pop [1,2] Peter Fritzson [3]

*Programming Environments Laboratory*
*Department of Computer and Information Science*
*Linköping University*
*Linköping, Sweden*

**Abstract**

The Structural Operational Semantics Development Tooling (SOSDT) Eclipse Plugin integrates the Relational Meta-Language (RML) compiler and debugger with the Eclipse Integrated Development Environment Framework. SOSDT, together with the RML compiler and debugger, provides an environment for developing and maintaining executable Structural Operational Semantics specifications, including the Natural Semantics big step variant of SOS specifications. The RML language is successfully used at our department for writing large specifications for a range of languages like Java, Modelica, Pascal, MiniML etc. The SOSDT environment includes support for browsing, code completion through menus or popups, code checking, automatic indentation, and debugging of specifications.

*Key words:* SOS, Natural Semantics, executable specification, Eclipse, RML, debugging.

## 1 Introduction

No programming language environment can be considered mature if is not supported by a strong set of tools which include execution, debugging, and profiling.

In this paper we present an integrated development environment called Structural Operational Semantics Development Tooling (SOSDT) [4] for

[1] This research was partially supported by the National Graduate School in Computer Science (CUGS) and the SSF RISE project.
[2] Email: adrpo@ida.liu.se
[3] Email: petfr@ida.liu.se

browsing, checking, and debugging semantic specifications. The SOSDT environment is based on the existing Relational Meta-Language (RML) system and its debugger and provides an easy to use graphical interface for these systems.

## 2 SOS/Natural Semantics and the Relational Meta-Language (RML)

Natural Semantics [2] is formalism for specifying many aspects of programming languages, e.g. type systems, dynamic semantics, translational semantics, static semantics, etc. Natural Semantics is an operational semantics derived from the Plotkin [6] structural operational semantics combined with the sequent calculus for natural deduction.

The Relational Meta-Language (RML) [5], is a practical language for writing executable SOS/Natural Semantics Specifications. The RML language is extensively used at out department for teaching and writing large specifications for different languages like Java, Modelica, MiniML, Pascal, etc. The RML language is compiled to highly efficient C code by the rml2c compiler. In this way, large parts of a compiler can be automatically generated from their Natural Semantics specifications. From the features of the RML language we can mention: strong static typing, simple module system, type inference, pattern matching and recursion are used for control flow, types can be polymorphic.

As pointed out in [3], the computer science community is constantly ignoring the debugging problem even though the debugging phase of software development takes more than the overall development time. Even if the RML language has a very short learning curve, the absence of debugging facilities previously created problems of understanding, debugging and verification of large specifications. We have addressed the debugging issue by providing a debugging framework for RML [7]. The debugger is based on abstract syntax tree instrumentation (program transformation) in the RML compiler and some runtime support. Type reconstruction is performed at runtime in order to present values of the user defined types.

## 3 The RML Integrated Environment (SOSDT) as an Eclipse Plugin

The SOSDT (previously named RML Development Tooling (RDT)) environment provides an integrated environment for our tools. The integrated environment with debugging and the various interactions between the components is presented in Figure 1 and 2.

The SOSDT environment has three major components, the RML Editor, the RML Browser and the RML Debugging components. All the components

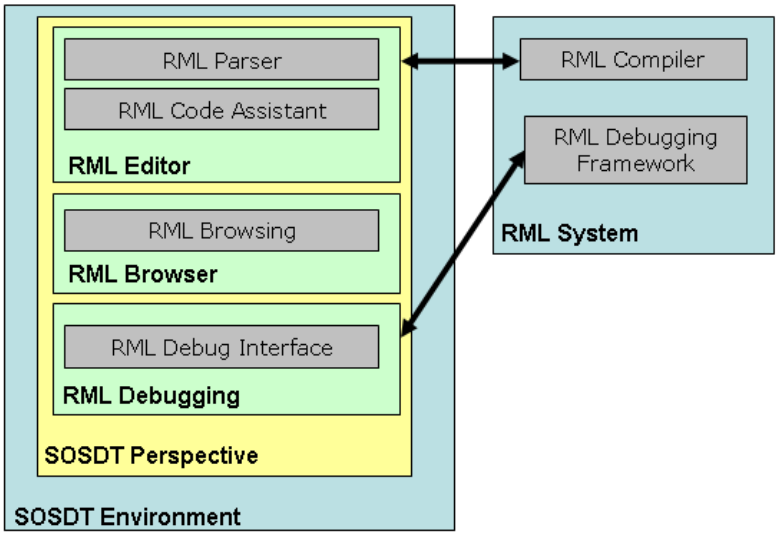



Fig. 1. Architecture of the RML system and SOSDT environment.

are active when the SOSDT perspective is selected within the Eclipse environment. Perspectives within Eclipse are used for configuration of views in connection with specific projects. Within the SOSDT environment the user creates and manages RML projects and RML files via wizards.

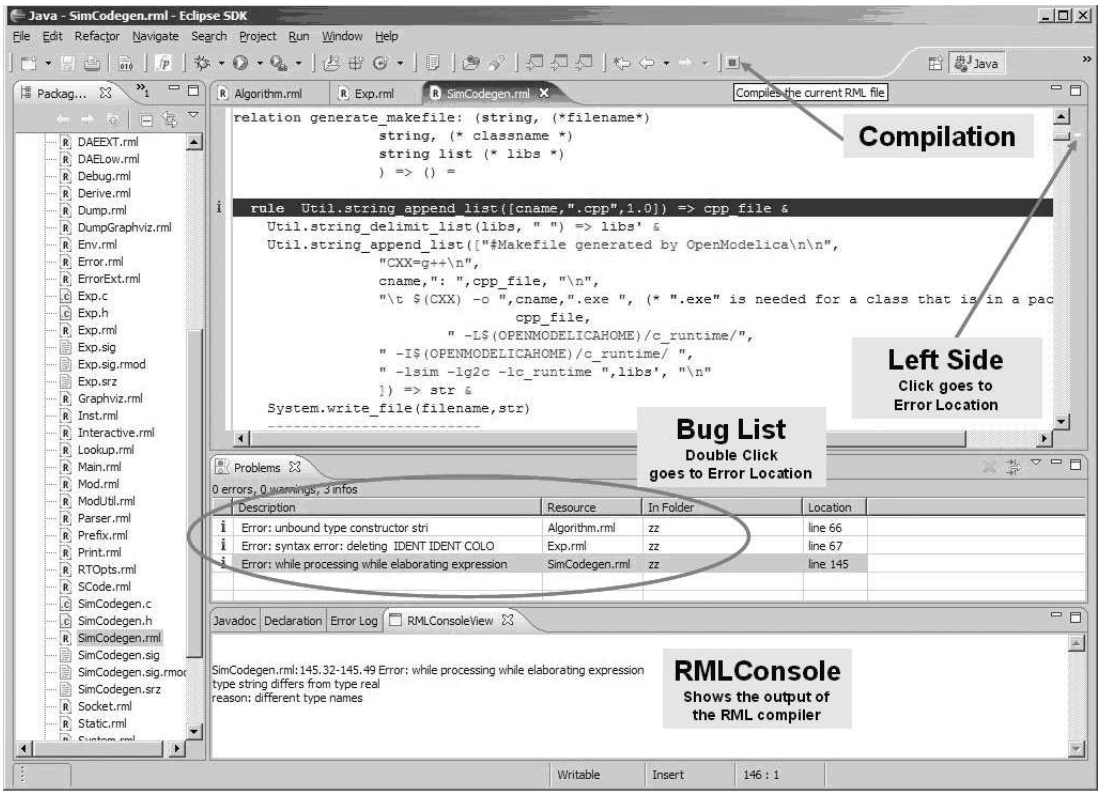



Fig. 2. SOSDT Eclipse Plugin for RML Development.

The RML Editor component provides syntax highlighting, auto indentation, code completion, type information and error highlighting. This component obtains the needed information from the RML parser and the RML Compiler. From the RML Compiler the errors and the type inference information is gathered. The type information is displayed when hoveding over a variable, relation or pattern. Code completion is provided when the user writes relation calls or patterns.

The RML Browser component provides easy navigation within an file. The RML parser is used to gather the information needed for browsing. The types, values, relations and rules are displayed within a tree for each RML file.

The RML Debugging component communicates via sockets with the RML Debugging Framework to provide debugging facilities like breakpoints, running and stepping, variable value inspection, etc.

All the SOSDT components are using the components of the Eclipse framework which are populated with information from the RML Parser and the RML Compiler. When a file is saved the RML Parser reads the file and updates the internal RML model information which triggers the update of the RML Browser. Also, on save the RML file is sent to the RML Compiler which dumps error information to be displayed in the Problems View and type information used to update the internal RML model.

## 4 Performance Evaluation

The test case used for the table below is based on an executable specification (SOS/Natural Semantics in RML) of the MiniFreja language [5] running a test program based on the sieve of Eratosthenes. All the needed information for reproducing the tests are available at http://www.ida.liu.se/~adrpo/sosdt/tests.

Mini-Freja is a call-by-name pure functional language. The test program calculates prime numbers. The Prolog translation (mf.pl) was originally implemented by Mikael Pettersson. The comparison was performed on a Fedora Core4 Linux machine with two AMD Athlon(TM) XP 1800+ processors at 1500 MHz and 1.5GB of memory. The measurements were done during April 2006.

| Prime# | RML | SICStus | SWI | Maude-MSOS-Tool |
|---|---|---|---|---|
| 8 | 0.00 | 0.05 | 0.00 | 2.92 |
| 30 | 0.02 | 1.42 | 1.79 | 226.77 |
| 40 | 0.06 | 3.48 | 3.879 | - |
| 50 | 0.13 | - | 11.339 | - |
| 100 | 1.25 | - | - | - |
| 200 | 16.32 | - | - | - |

Execution time is in seconds. The sign represents out of memory. The memory consumption was at peak 9Mb for RML. The other systems consumed

the entire 1.5Gb of memory and aborted at around 40 prime numbers. The largest executable specification developed so far using RML is the Modelica Language specification (an equation-based language), which is approximately 80 000 lines. We have improved compilation speed more than a factor of 10 since a year ago compiling 80 000 lines of RML now takes less than minute on a 1.5 GHz laptop.

## 5 Conclusions and Future Work

Our experience of writing large executable specifications in SOS/Natural Semantics style using RML for several different programming languages shows that a supportive development environment is essential also for developing specifications.

Therefore we have designed and implemented a prototype of an integrated environment for supporting such development, first as a version partly based on Emacs, and currently integrated in Eclipse [1], as an SOSDT Eclipse plugin. Some of our RML users who have debugged their specifications using a prototype of this environment have given us positive feedback and also various suggestions for improvement. While this is a good start, many improvements can be made to this environment. In the future we plan to improve the debugger execution speed, and implement additional features. Our goal is to provide a very well integrated and supportive development environment (IDE) for RML based on the Eclipse platform.

## References

[1] Eclipse Foundation, *Eclipse Development Platform*, http://www.eclipse.org.

[2] Gilles Kahn, *Natural Semantics*, Programming of Future Generation Computers. ed Niva M., p. 237.258, 1998.

[3] Henry Libermann, *The debugging scandal and what to do about it*, Communication of the ACM. vol:40(4), p:27-29, 1997.

[4] PELAB, *Structural Operational Semantics Development Tooling (SOSDT) Eclipse Plugin*, http://www.ida.liu.se/~adrpo/sosdt.

[5] Mikael Petterson, *Compiling Natural Semantics*, Ph.D. thesis, Linköping University, 1995, Dissertation No. 413,
also as Lecture Notes in Computer Science (LNCS) 1549, Springer-Verlag, 1999, RML Site: http://www.ida.liu.se/labs/pelab/rml.

[6] Gordon D. Plotkin, *A Structural Approach to Operational Semantics*, The Journal of Logic and Algebraic Programming 60-61 , 17-139., 2004.

[7] Adrian Pop and Peter Fritzson, *Debugging Natural Semantics Specifications*, Sixth International Symposium on Automated and Analysis-Driven Debugging (AADEBUG2005), September 19-21 2005, Monterey, California.

# Local bigraphs and confluence: two conjectures

## (extended abstract) ⋆

Robin Milner [1]

*University of Cambridge*

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

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

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

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

⋆ Appears also in the preliminary proceedings of the Express and Infinity workshops.

[1] Joint Express-Infinity-SOS Invited Speaker

*This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science*

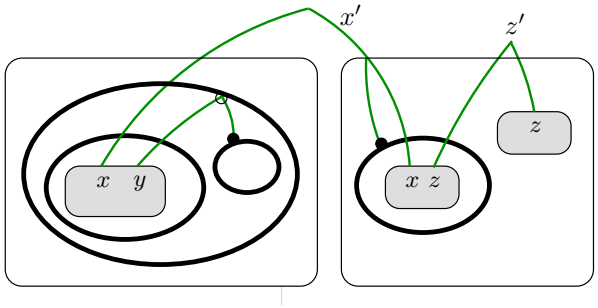



Fig. 1. A local bigraph $G : \langle\{xy\}, \{xz\}, \{z\}\rangle \to \langle\{x'\}, \{x'z'\}\rangle$

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

## 1 Local bigraphs

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

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

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

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

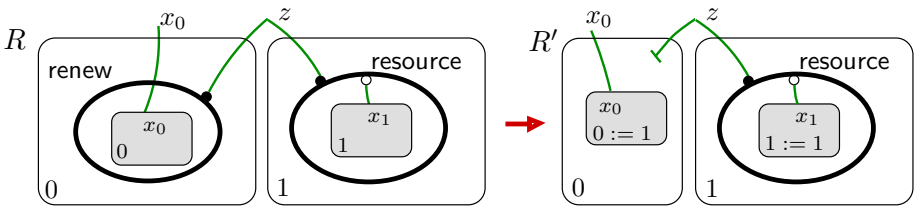



Fig. 2. A parametric reaction rule

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

An *agent* $a : \epsilon \to I$ has no sites; $\epsilon$ is the trivial interface with width 0. We use lower-case letters for agents.

## 2 Reaction rules and $\lambda$-calculus

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

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

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

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

$$M \quad ::= \quad x \mid \lambda x M \mid MN \mid M\,[x{:=}N]$$

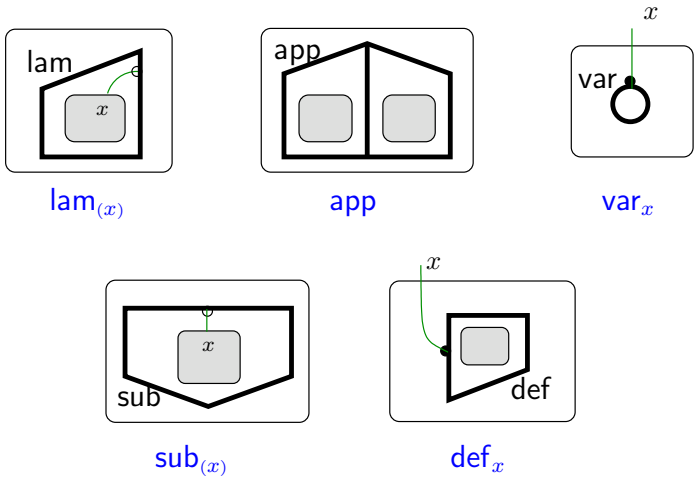



Fig. 3. Ions for the ˆΛBIG, with their algebraic representation

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

**Definition 2.1 (reduction)** The reduction rules in $\Lambda_{\mathsf{sub}}$ are as follows:

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

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

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

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

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

We now turn to ˆΛBIG, the BRS corresponding to $\Lambda_{\mathsf{sub}}$. Figure 3 shows its signature both graphically and algebraically. There are five controls (kinds of node), shown as *ions* (elementary bigraphs); a var-node has no sites, an app-node has two, and the rest have one. lam- and sub-nodes bind a link; var- and def-nodes have one port. The shapes of a node is unimportant, except that the shape of the app-node signifies that its sites are in left-to-right order.[2] Note that binding names are parenthesized.

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

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

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

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

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

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

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

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

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

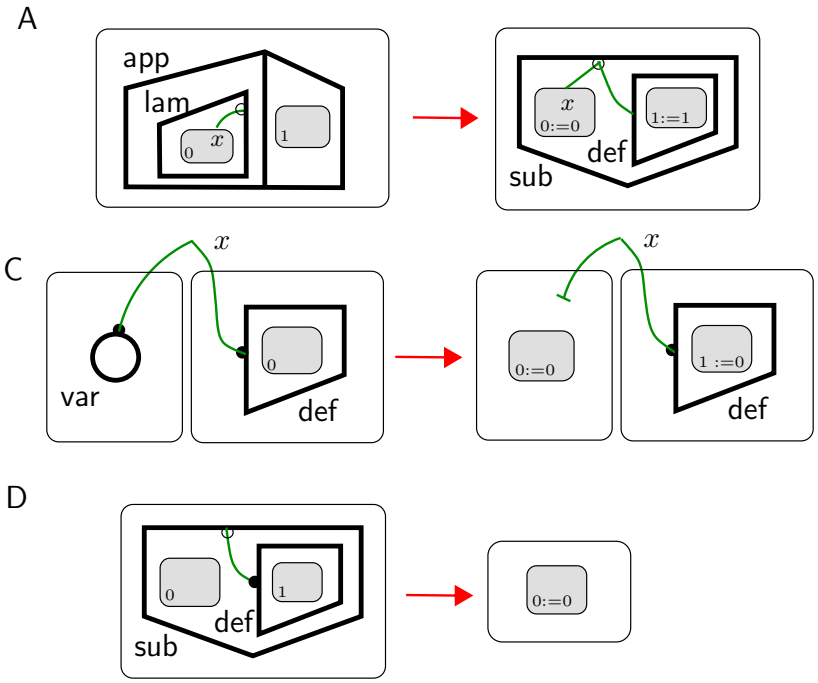
We now translate $\Lambda_{\mathsf{sub}}$ into $\hat{}\Lambda_{\text{BIG}}$. The translation function $[\![M]\!]_X$ is indexed by the set $X$, which must include all the free variables of $M$. Thus each term $M$ has many bigraph images. This technique was used to model the asynchronous $\pi$-calculus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

**Definition 2.2 ($\lambda$-terms into bigraphs)**

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

■

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




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

Fig. 4. Parametric reaction rules for $\hat{}\Lambda_{\mathrm{BIG}}$

**Definition 2.3 (dynamics)** $\hat{}\Lambda_{\mathrm{BIG}}$ has three reaction rules: A (apply), C (copy) and D (discard). They are shown both graphically and algebraically in Figure 4. ∎

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

We now assert that reaction in $\hat{}\Lambda_{\mathrm{BIG}}$ exactly matches reduction in $\Lambda_{\mathsf{sub}}$:

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

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

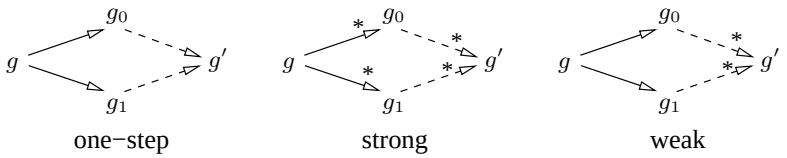



Fig. 5. Three notions of confluence

## 3 Confluence in bigraphs

Recall that for any given reduction or reaction relation '$\longrightarrow\!\!\!\!\triangleright$' there are three familiar notions of *confluence*, shown in Figure 5. They all say that if $g$ can react to become either $g_0$ or $g_1$, then these two reacta can in turn react to reach a common result. Clearly *one-step* $\Rightarrow$ *strong* $\Rightarrow$ *weak*, and it is well-known that these implications are strict in general. The most positive result for a BRS would be that strong confluence holds outright. This is indeed true (the Church-Rosser theorem) for the classical $\lambda$-calculus, and (by O'Conchuir) for $\Lambda_{\mathsf{sub}}$ also. However, in bigraphs we cannot expect this in general. Instead, we shall look for conditions that ensure non-interference between two competing reactions $g \longrightarrow\!\!\!\!\triangleright\, g_0$ and $g \longrightarrow\!\!\!\!\triangleright\, g_1$; such conditions may depend on the reaction rules that underlie the two translations, and on the extent to which the two redices overlap (if at all) in $g$. Moreover, it is in general easier to establish weak confluence in such cases.

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

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

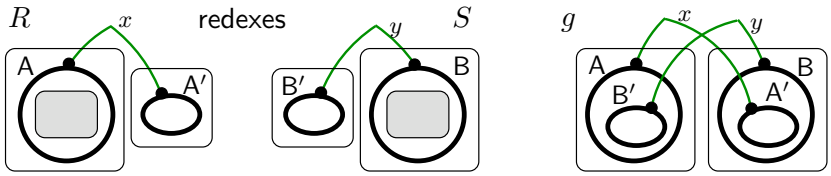



Fig. 6. An agent $g$ containing two intertwined redices $R$ and $S$

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

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

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

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

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

## Conclusion

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

## References

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

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

[3] Bundgaard, M. and Hildebrandt, T. (2006), Bigraphical semantics of higher-order mobile embedded resources with local names. Proc. GT-VC 2005,

Electronic Notes in Theoretical Computer Science 154(2), pp7–29.

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

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

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

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

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

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

# *Active* Evaluation Contexts for Reaction Semantics

Henrik Pilegaard Flemming Nielson Hanne Riis Nielson
{hepi,nielson,riis}@imm.dtu.dk

*Informatics and Mathematical Modelling*
*The Technical University of Denmark*

**Abstract**

In the context of process algebras it is customary to define semantics in the form of a reaction relation supported by a structural congruence relation. Recently process algebras have grown more expressive in order to meet the modelling demands of fields as diverse as business modelling and systems biology. This leads to combining various features, such as general choice and parallelism that were previously studied separately, and it often becomes difficult to define the reaction semantics. We present a general approach based on *active evaluation contexts* that allows the reaction semantics to be easily constructed.

*Key words:* Structural Operational Semantics, Reaction Semantics, Process Algebra, General Choice, Calculus of Communicating Systems, BioAmbients.

## 1 Introduction

Since their proposal [7,11,1] process calculi have become the primary tool for researching paradigms of concurrent computation. In the three decades that have passed two types of semantics have emerged:

**Structural operational semantics** [15] describes how processes may interact with their immediate environment. As usual for structural operational semantics the immediate behaviour of a composite process is defined structurally in terms of the immediate behaviours of its component processes.

Behaviours are often expressed using labelled transition systems. The label languages have considerable potential, which ensures that the structural operational semantics approach is viable for more expressive calculi also. As calculi do become more expressive, however, the required label languages tend to grow complicated and somewhat obscure the intuition of concurrency.

*This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science*

**Reaction semantics** in the style of the chemical abstract machine (CHAM) [2] clearly expresses an intuitive (chemical) understanding of concurrency. Every process term is perceived as the description of a solution of (syntactic) reactive entities. The usual *structural congruence* is a magical stirring mechanism that allows syntactic entities to float and mix as required. The *reaction relation* then simply states how reactions happen when 'matching' reactive entities come sufficiently close to each other.

While very intuitive this type of semantics has the drawback that the structural congruence is subject to conflicting requirements. On the one hand, we demand that the congruence sharply distinguishes between semantically different process expressions. On the other hand, we require it to be able to move potential redex constituents, which are syntactically located arbitrarily far apart, close enough together for a reaction rule to identify them. The usual decidability problems aside, these two requirements seem to clash as calculi grow in expressiveness.

Due to their different strengths it is usual for calculi to have both types of semantics. However, for the more elaborate calculi this is often difficult *unless* syntactic restrictions are imposed.

Milner's Calculus of Communicating Systems (CCS) is a prime example. The more recent version [13], which we briefly describe in Section 2, restricts choice to guarded sum. This facilitates both types of semantics because a normal form

$$\sum_j \alpha_j.P_j \tag{1}$$

can always be assumed for the constituents of redexes.

This may be contrasted to the original calculus [11] that has unrestricted choice. For this reason the substantially more complex normal form

$$(\ldots(((\alpha.P + P')|P'') + P''')|P'''' \ldots) \tag{2}$$

needs to be assumed for the constituents of redexes. This normal form is hard to match syntactically and the structural congruence is of little help as it is semantically meaningless to allow choice and parallel to distribute freely over one another. Thus, traditionally, only structural operational semantics is defined for derivatives of this calculus.

In this paper we show how reaction semantics can be defined even for very expressive calculi. One may ask why it is of interest to be able to deal with a binary unrestricted choice as opposed to an indexed guarded sum (over some arbitrary finite index set). In doing so we follow one of the design principles used by Gordon Plotkin when devising Structural Operational Semantics [15]: that one should always strive to use unary or binary syntactic constructors rather than general $n$-ary constructors because the former choice assists machine readable formal semantics and also gives a deeper semantic understanding of the programming construct at hand [16].

$$P ::= \mathbf{0} \mid \alpha \,.\, P \mid \sum_{i \in I} \alpha_i \,.\, P_i \mid P \mid P \mid (\nu a)\, P \qquad \alpha ::= \tau \mid a \mid \overline{a}$$

Fig. 1. Syntax of finite core CCS with guarded sums (CCSgs).

The proposed approach is based on a novel notion of *active evaluation contexts*. These contexts arise naturally when one allows standard evaluation contexts, originally proposed by Felleisen [5], to evolve when reactions occur. In Section 3 we develop the active evaluation contexts and use them to define a reaction semantics for the recursion-free fragment of CCS with unrestricted choice. The main theoretical result of this paper is that the resulting reaction semantics agrees with Milner's original structural operational semantics for closed expressions.

In the case of more complicated calculi the notions of active evaluation contexts and structural congruence combine nicely to give the desired semantics. We illustrate this in Section 4 where we define a reaction semantics for the full BioAmbients calculus extended with unrestricted choice.

## 2 CCS with Guarded Sums

In order to set the scene we start by considering CCS with guarded sums as defined by Milner [13]. In order to expose our contribution in Section 3 more clearly we shall focus on the finite fragment of the language; thus omitting recursion. This does not indicate a limitation in our framework - as we shall demonstrate later, in Section 4, recursion can easily be incorporated using a structural congruence.

Now, let $\mathcal{N}$, ranged over by $a, b, \cdots$, be a denumerable set of channel names and let the special symbol $\tau$ denote internal actions. The syntactical class of action prefixes, $\alpha \in \mathrm{Act}$, then contains all names $a \in \mathcal{N}$, all corresponding co-names $\overline{a} \in \mathcal{N}$, and the special symbol $\tau$. In this context the class of finite core CCS processes with guarded sums, to be denoted CCSgs, is described by the grammar in Figure 1, where we assume the $I$ in $\sum_{i \in I} \alpha_i \,.\, P_i$ to be finite and write $\mathbf{0}$ when $|I| = 0$ and $\alpha.P$ when $|I| = 1$.

Because the choice construct $\sum_{i \in I} \alpha_i \,.\, P_i$ is guarded it is possible to define a traditional (CHAM style) reaction semantics. As always, due to the syntactical nature of the reaction semantics, the definition relies on a structural congruence relation. If we let $\equiv_\alpha$ denote ordinary $\alpha$-equivalence, the structural congruence, $\equiv_{\mathrm{gs}}$, is the least relation that satisfies the axioms and rules in Figure 2. Using the congruence the reaction relation, $\longrightarrow_{\mathrm{gs}}$, defined by the axioms and rules of Figure 3 specifies the full reaction semantics of CCSgs. Note how this definition relies on the existence of the previously described normal forms of type (1).

Next we define a structural operational semantics specifying the process behaviour in terms of labelled transition systems. We assume the same class

Reordering of parallel processes:
$P \mid \mathbf{0} \equiv_{\mathsf{gs}} P$
$P \mid Q \equiv_{\mathsf{gs}} Q \mid P$
$P \mid (Q \mid R) \equiv_{\mathsf{gs}} (P \mid Q) \mid R$

Scope rules for name restrictions:
$(\nu a)\ \mathbf{0} \equiv_{\mathsf{gs}} \mathbf{0}$
$(\nu a)\,(\nu b)\,P \equiv_{\mathsf{gs}} (\nu b)\,(\nu a)\,P$
$(\nu a)\,(P \mid Q) \equiv_{\mathsf{gs}} P \mid (\nu a)\,Q \quad \text{if } a \notin \mathrm{fn}(P)$

Alpha equivalence:
$P \equiv_\alpha Q \Rightarrow P \equiv_{\mathsf{gs}} Q$

Reordering of term in a summation:
Summands can be freely reordered.

Equivalence:
$P \equiv_{\mathsf{gs}} P$
$P \equiv_{\mathsf{gs}} Q \Rightarrow Q \equiv_{\mathsf{gs}} P$
$P \equiv_{\mathsf{gs}} Q \wedge Q \equiv_{\mathsf{gs}} R \Rightarrow P \equiv_{\mathsf{gs}} R$

Congruence:
$P \equiv_{\mathsf{gs}} Q \Rightarrow \alpha.P + M \equiv_{\mathsf{gs}} \alpha.Q + M$
$P \equiv_{\mathsf{gs}} Q \Rightarrow (\nu a)\,P \equiv_{\mathsf{gs}} (\nu a)\,Q$
$P \equiv_{\mathsf{gs}} Q \Rightarrow P \mid R \equiv_{\mathsf{gs}} Q \mid R$
$P \equiv_{\mathsf{gs}} Q \Rightarrow R \mid P \equiv_{\mathsf{gs}} R \mid Q$

Fig. 2. Structural congruence of CCSgs.

$$\text{TAU:}\quad \tau\,.\,P + M \longrightarrow_{\mathsf{gs}} P \qquad \text{RES:}\quad \frac{P \longrightarrow_{\mathsf{gs}} P'}{(\nu a)\,P \longrightarrow_{\mathsf{gs}} (\nu a)\,P'} \qquad \text{PAR:}\quad \frac{P \longrightarrow_{\mathsf{gs}} P'}{P \mid Q \longrightarrow_{\mathsf{gs}} P' \mid Q}$$

$$\text{REACT:}\quad (a\,.\,P + M) \mid (\overline{a}\,.\,Q + N) \longrightarrow_{\mathsf{gs}} P \mid Q \qquad \text{STRUCT:}\quad \frac{P \equiv Q \quad Q \longrightarrow_{\mathsf{gs}} Q' \quad Q' \equiv P'}{P \longrightarrow_{\mathsf{gs}} P'}$$

Fig. 3. Reaction relation of CCSgs.

$$\text{SUM}_t:\quad M + \alpha\,.\,P + N \xrightarrow{\alpha}_{\mathsf{gs}} P \qquad \text{REACT}_t:\quad \frac{P \xrightarrow{\lambda}_{\mathsf{gs}} P' \quad Q \xrightarrow{\overline{\lambda}}_{\mathsf{gs}} Q'}{P \mid P \xrightarrow{\tau}_{\mathsf{gs}} P' \mid Q'}$$

$$\text{L-PAR}_t:\quad \frac{P \xrightarrow{\alpha}_{\mathsf{gs}} P'}{P \mid Q \xrightarrow{\alpha}_{\mathsf{gs}} P' \mid Q} \qquad \text{R-PAR}_t:\quad \frac{Q \xrightarrow{\alpha}_{\mathsf{gs}} Q'}{P \mid Q \xrightarrow{\alpha}_{\mathsf{gs}} P \mid Q'}$$

$$\text{RES}_t:\quad \frac{P \xrightarrow{\alpha}_{\mathsf{gs}} P'}{(\nu a)\,P \xrightarrow{\alpha}_{\mathsf{gs}} (\nu a)\,P'} \quad \text{if } \mathsf{n}(\alpha) \neq a$$

Fig. 4. Structural operational semantics of CCSgs.

$\alpha$ of action prefixes as before but use the abbreviation $\lambda$ to denote action prefixes that are not internal (i.e. $\lambda \in \mathrm{Act}\backslash\{\tau\}$); we shall write $\mathsf{n}(\alpha)$ to denote the base name of any action prefix. The transition relation $\xrightarrow{\alpha}_{\mathsf{uc}}$ defining the structural operational semantics is then the least relation satisfying the axioms and rules of Figure 4.

While these two formulations of the CCSgs semantics are often used for different purposes they are intended to express the same behaviour for closed process expressions. Thus the following result [13] is crucial:

**Theorem 2.1 (For CCSgs reaction agrees with $\tau$-transition)** *For any CCSgs process $P$ we have that $P \xrightarrow{\tau}_{\mathsf{gs}} \equiv P'$ if and only if $P \longrightarrow_{\mathsf{gs}} P'$.*

**Proof.** See Milner [13] Theorem 5.6 □

$$P ::= \mathbf{0} \mid \alpha\,.\,P \mid P + P \mid P \,\big|\, P \mid (\nu a)\,P$$

Fig. 5. Syntax of finite core CCS with unrestricted choice (CCSuc).

$$\text{PRE}_t: \ \alpha\,.\,P \xrightarrow{\alpha}_{\mathsf{uc}} P \qquad \text{L-PAR}_t: \ \frac{P \xrightarrow{\alpha}_{\mathsf{uc}} P'}{P \,\big|\, Q \xrightarrow{\alpha}_{\mathsf{uc}} P' \,\big|\, Q} \qquad \text{L-SUM}_t: \ \frac{P \xrightarrow{a}_{\mathsf{uc}} P'}{P + Q \xrightarrow{\alpha}_{\mathsf{uc}} P'}$$

$$\text{R-PAR}_t: \ \frac{Q \xrightarrow{\alpha}_{\mathsf{uc}} Q'}{P \,\big|\, Q \xrightarrow{\alpha}_{\mathsf{uc}} P \,\big|\, Q'} \qquad \text{R-SUM}_t: \ \frac{Q \xrightarrow{a}_{\mathsf{uc}} Q'}{P + Q \xrightarrow{\alpha}_{\mathsf{uc}} Q'}$$

$$\text{RES}_t: \ \frac{P \xrightarrow{\alpha}_{\mathsf{uc}} P'}{(\nu a)\,P \xrightarrow{\alpha}_{\mathsf{uc}} (\nu a)\,P'} \ \text{if } \mathsf{n}(\alpha) \neq a \qquad \text{REACT}_t: \ \frac{P \xrightarrow{\lambda}_{\mathsf{uc}} P' \quad Q \xrightarrow{\overline{\lambda}}_{\mathsf{uc}} Q'}{P \,\big|\, Q \xrightarrow{\tau}_{\mathsf{uc}} P' \,\big|\, Q'}$$

Fig. 6. Structural operational semantics of FcCSSuc.

$$\mathbb{C} ::= [\,] \mid (\nu a)\,\mathbb{C} \mid \mathbb{C} \,\big|\, P \mid P \,\big|\, \mathbb{C} \mid \mathbb{C} + P \mid P + \mathbb{C}$$

Fig. 7. The active evaluation contexts of CCSuc.

## 3 Active Evaluation Contexts for CCS

When the calculus is generalised to finite core CCS with *unrestricted choice* (CCSuc), as shown in Figure 5, the picture changes. Neither Milner nor other contributors have ever defined a classic (CHAM style) reaction semantics for a derivative of this language - and for good technical reasons. We believe that the technical means to deal with normal forms of type (2) have simply been lacking, and for this reason calculi descending from CCSuc are traditionally given only a structural operational semantics similar to the one shown in Figure 6 [12].

We shall now propose a semantics for CCSuc that retains the intuition of reaction semantics, but avoids the difficulties of previous approaches. For this purpose we shall introduce a notion of *active evaluation contexts* as defined in Figure 7. As usual for process/evaluation contexts, active evaluation contexts are process expressions with exactly one hole [5,13,8]. Contrary to ordinary contexts, however, we shall allow active contexts to evolve when reactive subprocesses occupying their hole engage in reactions.

To facilitate this we define the *context reduction relation* described in Figure 8. It specifies exactly what happens to contexts when reactive subprocesses engage in reaction. The reduction ability of contexts enables the compact and elegant definition of the reaction relation, $\longrightarrow$, shown in Figure 9. Here we use the auxiliary function $\mathsf{masked}(\mathbb{C})$ to determine the names and co-names that are restricted by a context $\mathbb{C}$. The function is given by:

$$\mathsf{masked}(\mathbb{C}) = \{\lambda \mid \textit{some } (\nu\lambda)\,\mathbb{C}' \textit{ occurs in } \mathbb{C}\}$$

In particular, $\mathsf{masked}((\nu a)\,\mathbb{C}) = \{a\} \cup \mathsf{masked}(\mathbb{C})$.

$$\text{EMP}_c: \quad [\,] \longrightarrow [\,] \qquad \text{NEW}_c: \quad \frac{\mathbb{C} \longrightarrow \mathbb{C}'}{(\nu a)\,\mathbb{C} \longrightarrow (\nu a)\,\mathbb{C}'} \qquad \text{L-PAR}_c: \quad \frac{\mathbb{C} \longrightarrow \mathbb{C}'}{\mathbb{C} \mid P \longrightarrow \mathbb{C}' \mid P}$$

$$\text{R-PAR}_c: \quad \frac{\mathbb{C} \longrightarrow \mathbb{C}'}{P \mid \mathbb{C} \longrightarrow P \mid \mathbb{C}'} \qquad \text{L-SUM}_c: \quad \frac{\mathbb{C} \longrightarrow \mathbb{C}'}{\mathbb{C} + P \longrightarrow \mathbb{C}'} \qquad \text{R-SUM}_c: \quad \frac{\mathbb{C} \longrightarrow \mathbb{C}'}{P + \mathbb{C} \longrightarrow \mathbb{C}'}$$

Fig. 8. Context reduction for active evaluation contexts in CCSuc.

$$\text{TAU:} \quad \tau\,.\,P \longrightarrow_{\mathsf{uc}} P$$

$$\text{REACT:} \quad \frac{\mathbb{C}_1 \longrightarrow \mathbb{C}_1' \quad \mathbb{C}_2 \longrightarrow \mathbb{C}_2'}{\mathbb{C}_1[\,\lambda\,.\,P\,] \mid \mathbb{C}_2[\,\overline{\lambda}\,.\,Q\,] \longrightarrow_{\mathsf{uc}} \mathbb{C}_1'[\,P\,] \mid \mathbb{C}_2'[\,Q\,]} \text{ if } \mathsf{n}(\lambda) \notin (\mathsf{masked}(\mathbb{C}_1) \cup \mathsf{masked}(\mathbb{C}_2))$$

$$\text{CONT:} \quad \frac{\mathbb{C} \longrightarrow \mathbb{C}' \quad P \longrightarrow_{\mathsf{uc}} P'}{\mathbb{C}[\,P\,] \longrightarrow_{\mathsf{uc}} \mathbb{C}'[\,P'\,]}$$

Fig. 9. Reaction relation of CCSuc.

### *3.1 Correspondence of Semantics*

It is evident that that the context reduction relation strongly resembles those rules of the structural operational semantics that encode the recursive descent into process terms. Consequently, structural congruence turns out to be unnecessary as was the case for the structural operational semantics.

In this favourable context the equivalence of the reaction semantics and the structural operational semantics for closed process expressions can be expressed simply as:

**Theorem 3.1 (Reaction corresponds to $\tau$ transition)** *$P \longrightarrow_{\mathsf{uc}} P'$ if and only if $P \xrightarrow{\tau}_{\mathsf{uc}} P'$.*

The proof has two parts, but first we establish the following useful result, which shows how the notion of active contexts relates to the structural operational semantics:

**Lemma 3.2 (Contexts respect behaviour)** *If $P \xrightarrow{\alpha}_{\mathsf{uc}} P'$, $\mathbb{C} \longrightarrow \mathbb{C}'$, and $\mathsf{n}(\alpha) \notin \mathsf{masked}(\mathbb{C})$ then $\mathbb{C}[\,P\,] \xrightarrow{\alpha}_{\mathsf{uc}} \mathbb{C}'[\,P'\,]$.*

**Proof.** The proof proceeds by structural induction on $\mathbb{C}$:

Base case $[\,]$: trivial.

Case $\mathbb{C} \mid R$:

From the premises we have $P \xrightarrow{\alpha}_{\mathsf{uc}} P'$, $\mathbb{C} \mid R \longrightarrow (\mathbb{C} \mid R)'$, and $\mathsf{n}(\alpha) \notin \mathsf{masked}(\mathbb{C})$.
By the shape of the inference of $\longrightarrow$ we have $(\mathbb{C} \mid R)' = \mathbb{C}' \mid R$ and $\mathbb{C} \longrightarrow \mathbb{C}'$ as a necessary premise. From the induction hypothesis it is now clear that $\mathbb{C}[\,P\,] \xrightarrow{\alpha}_{\mathsf{uc}} \mathbb{C}'[\,P'\,]$.
A single application of the L-PAR$_t$ rule now establishes the desired result: $\mathbb{C}[\,P\,] \mid R \xrightarrow{\alpha}_{\mathsf{uc}} \mathbb{C}'[\,P'\,] \mid R$

Cases $R \mid \mathbb{C}$, $R + \mathbb{C}$, and $\mathbb{C} + R$:

All similar.

Case $(\nu a)\,\mathbb{C}$:

From the premises we have $P \xrightarrow{\alpha}_{\mathsf{uc}} P'$, $(\nu a)\,\mathbb{C} \longrightarrow ((\nu a)\,\mathbb{C})'$, and $\mathsf{n}(\alpha) \notin \mathsf{masked}((\nu a)\,\mathbb{C})$.
By the shape of the inference of $\longrightarrow$ we have $((\nu a)\,\mathbb{C})' = (\nu a)\,\mathbb{C}'$ and $\mathcal{C} \longrightarrow \mathcal{C}'$ as a necessary premise and we know that $\mathsf{n}(\alpha) \notin \mathsf{masked}(\mathbb{C})$.
From the induction hypothesis it is now clear that $\mathbb{C}[\,P\,] \xrightarrow{\alpha}_{\mathsf{uc}} \mathbb{C}'[\,P'\,]$.
Given that $\mathsf{n}(\alpha) \notin \mathsf{masked}((\nu a)\,\mathbb{C})$ we have $\mathsf{n}(\alpha) \neq a$ and a single application of the $\mathrm{RES}_t$ rule now establishes the desired result: $(\nu a)\,\mathbb{C}[\,P\,] \xrightarrow{\alpha}_{\mathsf{uc}} (\nu a)\,\mathbb{C}'[\,P'\,]$. □

Given this lemma it is now easy to establish the 'if' part of Theorem 3.1:

**Lemma 3.3** *If $P \longrightarrow_{\mathsf{uc}} P'$ then $P \xrightarrow{\tau}_{\mathsf{uc}} P'$.*

**Proof.** We proceed by induction on the inference of $\longrightarrow_{\mathsf{uc}}$:
Case TAU:

Given the process term $\tau.P$ rule $\mathrm{PRE}_t$ trivially instantiates to give us $\tau.P \xrightarrow{\tau}_{\mathsf{uc}} P$, just as desired.

Case REACT:

Rule $\mathrm{PRE}_t$ gives us $a\,.\,P_1 \xrightarrow{a}_{\mathsf{uc}} P_1$ and $\overline{a}\,.\,P_2 \xrightarrow{\overline{a}}_{\mathsf{uc}} P_2$, and from the rule premises we have that $\mathbb{C}_1 \longrightarrow \mathbb{C}_1'$, $\mathbb{C}_2 \longrightarrow \mathbb{C}_2'$, $\mathsf{n}(a) \notin (\mathsf{masked}(\mathbb{C}_1) \cup \mathsf{masked}(\mathbb{C}_2))$.
Using Lemma 3.2 we can establish that $\mathbb{C}_1[\,a\,.\,P_1\,] \xrightarrow{a}_{\mathsf{uc}} \mathbb{C}_1'[\,P_1\,]$ and $\mathbb{C}_2[\,\overline{a}\,.\,P_2\,] \xrightarrow{\overline{a}}_{\mathsf{uc}} \mathbb{C}_2'[\,P_2\,]$. By a single application of rule $\mathrm{REACT}_t$ we can now conclude $\mathbb{C}_1[\,a\,.\,P_1\,] \mid \mathbb{C}_2[\,\overline{a}\,.\,P_2\,] \xrightarrow{\tau}_{\mathsf{uc}} \mathbb{C}_1'[\,P_1\,] \mid \mathbb{C}_2'[\,P_2\,]$, as required.

Case CONT:

From the premises we have $\mathbb{C} \longrightarrow \mathbb{C}'$ and $P \longrightarrow_{\mathsf{uc}} P'$. Using the induction hypothesis on the latter we obtain $P \xrightarrow{\tau}_{\mathsf{uc}} P'$, where obviously $\mathsf{n}(\tau) \notin \mathsf{masked}(\mathbb{C})$.
Lemma 3.2 now tells us that $\mathbb{C}[\,P\,] \xrightarrow{\tau}_{\mathsf{uc}} \mathbb{C}'[\,P'\,]$, as required. □

We now turn to the 'only if' part of Theorem 3.1, which is a straightforward corollary of the following lemma:

**Lemma 3.4** *If $P \xrightarrow{\tau}_{\mathsf{uc}} Q$ then $P \longrightarrow_{\mathsf{uc}} Q$ and if $P \xrightarrow{\lambda}_{\mathsf{uc}} Q$ then, for all contexts $\mathbb{C}, \mathbb{C}'$ such that $\mathbb{C} \longrightarrow \mathbb{C}'$ and $\mathsf{n}(\lambda) \notin \mathsf{masked}(\mathbb{C})$, we have $\mathbb{C}[\,P\,] \mid \overline{\lambda}\,.\,R \longrightarrow_{\mathsf{uc}} \mathbb{C}'[\,Q\,] \mid R$ and $\overline{\lambda}\,.\,R \mid \mathbb{C}[\,P\,] \longrightarrow_{\mathsf{uc}} R \mid \mathbb{C}'[\,Q\,]$.*

**Proof.** The proof proceeds by induction on the inference of $\xrightarrow{\alpha}_{\mathsf{uc}}$:
Base case $\mathrm{PRE}_t$ :

If $\alpha$ is $\tau$ then $P$ is $\tau.P'$ and $Q$ is $P'$ and the transition $P \longrightarrow_{\mathsf{uc}} Q$ follows from TAU.
Otherwise $P$ is $\lambda.P'$ and $Q$ is $P'$ and the required transitions both follow from REACT (taking one of $\mathbb{C}_1$ and $\mathbb{C}_2$ to be $\mathbb{C}$ and the other to be $[\,]$).

Case L-PAR$_t$ :

If $\alpha$ is $\tau$ then $P$ is $P' \,|\, Q'$ and $Q$ is $P'' \,|\, Q'$ and the transition $P \longrightarrow_{\mathsf{uc}} Q$ follows from the induction hypothesis and CONT where $\mathbb{C}$ is taken to be $[\,] \,|\, Q'$.
Otherwise, $P$ and $Q$ are of a similar form, but the transitions have to follow from REACT. From premises we know that $P' \xrightarrow{\lambda}_{\mathsf{uc}} P''$, and the induction hypothesis then tells us that $\mathbb{C}[\,P'\,] \,|\, \overline{\lambda}\,.\,R \longrightarrow_{\mathsf{uc}} \mathbb{C}'[\,P''\,] \,|\, R$ for all suitable $\mathbb{C}, \mathbb{C}'$ (i.e. $\mathbb{C} \longrightarrow \mathbb{C}'$ with $\mathsf{n}(\lambda) \notin \mathsf{masked}(\mathbb{C})$). Given that $\mathbb{C}, \mathbb{C}'$ are suitable clearly $\mathbb{C}[\,[\,] \,|\, Q'\,], \mathbb{C}'[\,[\,] \,|\, Q'\,]$ are also suitable, and then the required transitions both follow from REACT.

Case R-PAR$_t$, LSUM$_t$, and R-SUM$_t$ :

All similar.

Case RES$_t$ :

If $\alpha$ is $\tau$ then $P$ is $(\nu a)\,P'$ and $Q$ is $(\nu a)\,P''$ and the transition $P \longrightarrow_{\mathsf{uc}} Q$ follows from the induction hypothesis and CONT where $\mathbb{C}$ is taken to be $(\nu a)\,[\,]$.
Otherwise, $P$ and $Q$ are of a similar form, but the transitions have to follow from REACT. From the premise we know that $P' \xrightarrow{\lambda}_{\mathsf{uc}} P''$, and the induction hypothesis then tells us that $\mathbb{C}[\,P'\,] \,|\, \overline{\lambda}\,.\,R \longrightarrow_{\mathsf{uc}} \mathbb{C}'[\,P''\,] \,|\, R$ for all suitable $\mathbb{C}, \mathbb{C}'$ (i.e. $\mathbb{C} \longrightarrow \mathbb{C}'$ with $\mathsf{n}(\lambda) \notin \mathsf{masked}(\mathbb{C})$). Given that $\mathbb{C}, \mathbb{C}'$ are suitable clearly $\mathbb{C}[\,(\nu a)\,[\,]\,], \mathbb{C}'[\,(\nu a)\,[\,]\,]$ are also suitable because we know from the side-condition that $\mathsf{n}(\lambda) \neq a$, which means that $\mathsf{n}(\lambda) \notin \mathsf{masked}(\mathbb{C}) \cup \{a\}$. The required transitions then both follow from REACT.

Case REACT$_t$ :

Here $\alpha$ is $\tau$ and $P$ is $P' \,|\, Q'$ and $Q$ is $P'' \,|\, Q''$. By the premises and the induction hypothesis we have both $\mathbb{C}[\,P'\,] \,|\, \overline{\lambda}\,.\,R \longrightarrow_{\mathsf{uc}} \mathbb{C}'[\,P''\,] \,|\, R$ and $\overline{\lambda}\,.\,R \,|\, \mathbb{C}[\,P'\,] \longrightarrow_{\mathsf{uc}} R \,|\, \mathbb{C}'[\,P''\,]$. As these reactions can only arise by the use of REACT we may assume that $\mathbb{C}[\,P'\,]$ is of the form $\mathbb{C}[\,\mathbb{C}_{\mathsf{deep}}[\,\lambda\,.\,P_{\mathsf{deep}}\,]\,]$, where $\mathbb{C}_{\mathsf{deep}} \longrightarrow \mathbb{C}'_{\mathsf{deep}}$ and $\mathsf{n}(\lambda) \notin \mathsf{masked}(\mathbb{C}_{\mathsf{deep}})$. Similar arguments for $Q'$ allows us to assume that $\mathbb{C}[\,Q'\,]$ is of the form $\mathbb{C}[\,\mathbb{C}_{\mathsf{deep}}[\,\overline{\lambda}\,.\,Q_{\mathsf{deep}}\,]\,]$, where $\mathbb{C}_{\mathsf{deep}} \longrightarrow \mathbb{C}'_{\mathsf{deep}}$ and $\mathsf{n}(\lambda) \notin \mathsf{masked}(\mathbb{C}_{\mathsf{deep}})$. From this we obtain the desired reaction by a single application of REACT. □

## 4 Active Evaluation Contexts for BioAmbients

The use of process calculi as modelling languages for real-world domains, such as business modelling and systems biology, seems to be a current trend in language based technology. The trend combines many language features that were previously unstudied or only studied in isolation. This invariably leads to evermore expressive calculi that share the difficulties of CCSuc with respect to the definition of appropriate reaction semantics.

The BioAmbients calculus of Regev et al. [18,17,3] is a prime example. The language is a sibling of Mobile Ambients (Cardelli and Gordon [4]) designed to model biological systems. It preserves the notion of ambients as bounded mobile sites of activity; contrary to Mobile Ambients, however, bio-ambients are cast as nameless entities. The ambients are used to model chemically active sub-systems (compartments) bound by biological barriers (membranes) in an intuitive manner.

The calculus is quite extensive in terms of modelling primitives. Appropriate sets of capabilities and co-capabilities are devised for modelling a variety of biological reactions, such as movement and communication, that may happen between the sub-systems. Both communication and movement are facilitated by having capability/co-capability pairs react with each other as in [10,14]. As a consequence all reactions are synchronous in the sense that the process exposing the capability and the process exposing the corresponding co-capability must simultaneously agree on a reaction for it to happen. Such an agreement can be reached only if the two entities share the same (channel) name.

The set of control structures for processes is slightly larger than what is traditionally studied for Mobile Ambients. Besides the ambient construct it includes non-deterministic (external) choice as well as a general recursion construct in the manner of CCS [12] in order to facilitate the description of more faithful models of biological systems.

Following the tradition of ambient calculi BioAmbients is endowed by Regev with a (CHAM style) reaction semantics [18,17]. Arguably, this is a natural choice because it ensures a high degree of coherence between the inherently bio-chemical modelling domain and the operational model of the language. As for CCSgs, however, external choice is limited to guarded sums and, again, we believe that this is so because the technical means to combine parallelism and unrestricted choice was lacking at the time of definition.

In the following we present a BioAmbients variant where choice is unrestricted. We trust this to be a conservative extension of the original calculus, but a formal proof is besides the point of the present paper. Rather, we shall focus on defining a reaction semantics using our active evaluation contexts.

### *4.1 Syntax*

The full syntax of BioAmbients is defined in Figure 10. Note that we use the heavy brackets ⟦ and ⟧ to represent ambient boundaries; the ordinary brackets [

| | | |
|---|---|---|
| $P$ | $::= \mathbf{0}$ | a terminal (stuck) process |
| | $\mid (\nu a)\, P$ | restricting the scope of $a$ to the process expression $P$ |
| | $\mid \lfloor\, P \,\rfloor$ | process $P$ enclosed by ambient boundary |
| | $\mid P \mid P'$ | process $P$ in parallel with process $P'$ |
| | $\mid P + P'$ | non-deterministic external choice between $P$ and $P'$ |
| | $\mid M\,.\,P$ | capability prefixed process |
| | $\mid \mathsf{rec}\, X.\, P$ | recursive process definition ($X = P$) |
| | $\mid X$ | process identifier |
| $M$ | $::= \mathsf{enter}\ a \mid \mathsf{accept}\ a$ | enter movement |
| | $\mid \mathsf{exit}\ a \mid \mathsf{expel}\ a$ | exit movement |
| | $\mid \mathsf{merge}\text{–}\ a \mid \mathsf{merge}\text{+}\ a$ | merge movement |
| | $\mid a!\{b\} \mid a?\{c\}$ | local communication binding the variable $c$ |
| | $\mid a\_!\{b\} \mid a\hat{}?\{c\}$ | parent to child communication binding the variable $c$ |
| | $\mid a\hat{}!\{b\} \mid a\_?\{c\}$ | child to parent communication binding the variable $c$ |
| | $\mid a\#!\{b\} \mid a\#?\{c\}$ | sibling communication binding the variable $c$ |

Fig. 10. Syntax of BioAmbients.

$$\mathbb{C} ::= [\,] \mid \mathbb{C} \mid P \mid P \mid \mathbb{C} \mid \mathbb{C} + P \mid P + \mathbb{C}$$

Fig. 11. The active evaluation contexts of BioAmbients.

and ] are reserved for substitutions and holes of contexts. We use $a, b, \cdots \in \mathcal{N}$ to denote channel names and $M \in \mathbf{Cap}$ for the notion of *(co-) capabilities*, which are based on names and generalise the notion of actions. As customary for BioAmbients we omit the notion of internal $\tau$-actions. Also, since reactions are based on (co-)capabilities, we have no need for co-names.

In the following we shall write $P[a/b]$ to denote the process that is as $P$ except that all free occurrences of the name $b$ are replaced by $a$. Similarly, we shall use $P[Q/X]$ to identify the process that is as $P$ except that all free occurrences of the process identifier $X$ are replaced by the process expression $Q$. In both cases we take care to perform the necessary $\alpha$-renamings to avoid capturing free names and process identifiers. Finally, we shall use $\mathsf{fn}(P)$ to pick out the free names of a process $P$ and write $P \equiv_\alpha Q$ to state that two processes $P$ and $Q$ are identical up to $\alpha$-renaming of names.

### *4.2 Semantics*

The active evaluation contexts of BioAmbients, shown in Figure 11 and Figure 12, are simpler than those of CCSuc. Their definition embodies three crucial choices, which we shall further substantiate below:

**(i)** The active contexts are *name restriction free.*

**(ii)** The active contexts are *ambient boundary free.*

$$\text{EMP}_c : \quad [\,] \longrightarrow [\,] \qquad \text{L-PAR}_c : \quad \frac{\mathbb{C} \longrightarrow \mathbb{C}'}{\mathbb{C} \mid P \longrightarrow \mathbb{C}' \mid P} \qquad \text{R-PAR}_c : \quad \frac{\mathbb{C} \longrightarrow \mathbb{C}'}{P \mid \mathbb{C} \longrightarrow P \mid \mathbb{C}'}$$

$$\text{L-SUM}_c : \quad \frac{\mathbb{C} \longrightarrow \mathbb{C}'}{\mathbb{C} + P \longrightarrow \mathbb{C}'} \qquad \text{R-SUM}_c : \quad \frac{\mathbb{C} \longrightarrow \mathbb{C}'}{P + \mathbb{C} \longrightarrow \mathbb{C}'}$$

Fig. 12. Reduction of BioAmbients active evaluation contexts.

**(iii)** The active contexts are *recursion free*.

The choice **(i)** is necessary because both $\pi$-style name passing and ambient style movement may cause extrusion of scope. This happens when restricted names are communicated to recipients or moved to positions outside of their original bounding box. Defining the active contexts to be name restriction free allows us to deal explicitly with all scope related issues in the usual way, i.e. using the structural congruence, shown in Figure 13, to migrate name restrictions in and out of redexes as required.

Contrary to the usual practice we allow constant introductions ($\nu a$) to migrate in and out of non-deterministic external choice constructs in much the same way as is customary for parallel composition. This is necessary because the rules of our reaction semantics are implicitly going to assume the normal form

$$(\ldots(((\lfloor\,(\ldots(((\mathbf{M}\,.\,P_i + P_i') \mid P_i'') + P_i''') \mid P_i'''' \ldots)\,\rfloor + P_o') \mid P_o'') + P_o''') \mid P_o'''' \ldots) \quad (3)$$

for the constituents of redexes of movement actions, and

$$(\ldots(((\lfloor(\ldots(((\mathbf{M}\,.\,P_i + P_i') \mid P_i'') + P_i''') \mid P_i'''' \ldots)\rfloor + P_o') \mid P_o'') + P_o''') \mid P_o'''' \ldots) \quad (4)$$

(where the grey symbols denote syntax that may, or may not, be present) for the constituents of redexes of communication actions. In each of these cases the congruence must be strong enough to migrate an obstructing name restriction out of the way, if appropriate.

The choice **(ii)** is required to ensure that rules of the reaction semantics, shown in Figure 14, recognise and alter redexes correctly. All redexes have two constituents, one exposing a capability prefix and another exposing the corresponding co-capability prefix. As mentioned, these constituents can always be assumed to be of one of the forms (3) or (4), which implies that there are some cases where exactly one boundary is demanded to enclose the exposed prefix, and other cases where no boundaries are allowed. Defining the active evaluation contexts to be ambient boundary free allows us to easily match each of these cases in the following manner:

**(I)** If no ambient boundary is allowed, the constituent is simply a capability prefixed process expression enclosed in an active evaluation context, which we match by $\mathbb{C}[\,\mathbf{M}\,.\,P\,]$.

**(II)** If exactly one ambient boundary is demanded, the constituent is an expression of the form **(I)** enclosed in an ambient boundary construct and a

Scope rules for namebindings:

$$(\nu a)\,\mathbf{0} \equiv \mathbf{0} \qquad (\nu a)\,(P \mid P') \equiv ((\nu a)\,P) \mid P' \quad \text{if } a \notin \mathsf{fn}(P')$$

$$(\nu a_1)\,(\nu a_2)\,P \equiv (\nu a_2)\,(\nu a_1)\,P \qquad (\nu a)\,(P + P') \equiv ((\nu a)\,P) + P' \quad \text{if } a \notin \mathsf{fn}(P')$$

$$(\nu a)\,(\lfloor\!\lfloor\ P\ \rfloor\!\rfloor) \equiv \lfloor\!\lfloor\ (\nu a)\,P\ \rfloor\!\rfloor$$

Unfolding of recursion:

$$\mathsf{rec}\,X.\,P \equiv P[\mathsf{rec}\,X.\,P/X]$$

$\alpha$-renaming:

$$\frac{P \equiv_\alpha Q}{P \equiv Q}$$

Congruence requirements:

$$P \equiv P \qquad \frac{P \equiv Q}{Q \equiv P} \qquad \frac{P \equiv Q \quad Q \equiv R}{P \equiv R}$$

$$\frac{P \equiv Q}{\mathbb{C}[\,P\,] \equiv \mathbb{C}[\,Q\,]} \qquad \frac{P \equiv Q}{\lfloor\!\lfloor\ P\ \rfloor\!\rfloor \equiv \lfloor\!\lfloor\ Q\ \rfloor\!\rfloor} \qquad \frac{P \equiv Q}{(\nu a)\,P \equiv (\nu a)\,Q}$$

Fig. 13. Structural congruence $P \equiv Q$ for BioAmbients.

further active evaluation context, which we match by $\mathbb{C}_1\lfloor\!\lfloor\ \mathbb{C}_2[\,\mathbf{M}\,.\,P\,]\ \rfloor\!\rfloor$.

As illustrated by Figure 14, where the active contexts are toned down, systematic application of these patterns allows us to focus entirely on the high level structure of redexes and contractums while the contexts conveniently hide the details of redex constituents as well as reactions.

Finally, the choice **(iii)** completely separates the notion of recursion from that of the active evaluation contexts. As a result recursion is easily handled in the usual manner, i.e. using the structural congruence to unfold recursive processes as required.

## 5 Related Work

Employing evaluation contexts to express semantics of process calculi is not a new idea.

Berry and Boudol [2] use *program contexts* to denote the arbitrary testing environments that form the basis of semantic equivalence in CHAM. Later authors, such as Milner [13], use a similar (derived) notion of *process contexts*, primarily in order to extend equivalences to congruences. A few authors, such as Godskesen, Hildebrandt, and Sasone [6] for the Calculus of Mobile Resources, also use similar derived notions (*path contexts*, *evaluation contexts*, *resource contexts* etc.) to define the actual reaction relation of their calculi. In all cases, however, the involved notions of context are (standard) static ones and none of the authors address the issue of combining general choice with parallelism.

Sewell [20] makes a radically different use of contexts. He shows how to automatically derive labelled transition systems from a variety of rewrite semantics by simply using suitable contexts as transition labels whenever reaction occurs. This allows operational equivalences, as provided by the reaction

**Movement of ambients:**

$$\frac{\mathbb{C}_1 \longrightarrow \mathbb{C}_1' \quad \mathbb{C}_2 \longrightarrow \mathbb{C}_2' \quad \mathbb{C}_3 \longrightarrow \mathbb{C}_3' \quad \mathbb{C}_4 \longrightarrow \mathbb{C}_4'}{\mathbb{C}_1\lfloor\, \mathbb{C}_2[\,\mathsf{enter}\ a\,.\,P\,]\,\rfloor \,\big|\, \mathbb{C}_3\lfloor\, \mathbb{C}_4[\,\mathsf{accept}\ a\,.\,Q\,]\,\rfloor \longrightarrow \mathbb{C}_1'[\,\mathbf{0}\,] \,\big|\, \mathbb{C}_3'\lfloor\, \mathbb{C}_2'[\,P\,]\,\rfloor \,\big|\, \mathbb{C}_4'[\,Q\,]\,\rfloor}$$

$$\frac{\mathbb{C}_1 \longrightarrow \mathbb{C}_1' \quad \mathbb{C}_2 \longrightarrow \mathbb{C}_2' \quad \mathbb{C}_3 \longrightarrow \mathbb{C}_3'}{\lfloor\, \mathbb{C}_1\lfloor\, \mathbb{C}_2[\,\mathsf{exit}\ a\,.\,P\,]\,\rfloor \,\big|\, \mathbb{C}_3[\,\mathsf{expel}\ a\,.\,Q\,]\,\rfloor \longrightarrow \lfloor\, \mathbb{C}_2'[\,P\,]\,\rfloor \,\big|\, \mathbb{C}_1'[\,\mathbf{0}\,] \,\big|\, \mathbb{C}_3'[\,Q\,]\,\rfloor}$$

$$\frac{\mathbb{C}_1 \longrightarrow \mathbb{C}_1' \quad \mathbb{C}_2 \longrightarrow \mathbb{C}_2' \quad \mathbb{C}_3 \longrightarrow \mathbb{C}_3' \quad \mathbb{C}_4 \longrightarrow \mathbb{C}_4'}{\mathbb{C}_1\lfloor\, \mathbb{C}_2[\,\mathsf{merge}\text{–}\ a\,.\,P\,]\,\rfloor \,\big|\, \mathbb{C}_3\lfloor\, \mathbb{C}_4[\,\mathsf{merge}\text{+}\ a\,.\,Q\,]\,\rfloor \longrightarrow \mathbb{C}_1'[\,\mathbf{0}\,] \,\big|\, \mathbb{C}_3'\lfloor\, \mathbb{C}_2'[\,P\,] \,\big|\, \mathbb{C}_4'[\,Q\,]\,\rfloor}$$

**Communication between ambients:**

$$\frac{\mathbb{C}_1 \longrightarrow \mathbb{C}_1' \quad \mathbb{C}_2 \longrightarrow \mathbb{C}_2'}{\mathbb{C}_1[\,a!\{b\}\,.\,P\,] \,\big|\, \mathbb{C}_2[\,a?\{c\}\,.\,Q\,] \longrightarrow \mathbb{C}_1'[\,P\,] \,\big|\, \mathbb{C}_2'[\,Q[m/p]\,]}$$

$$\frac{\mathbb{C}_1 \longrightarrow \mathbb{C}_1' \quad \mathbb{C}_2 \longrightarrow \mathbb{C}_2' \quad \mathbb{C}_3 \longrightarrow \mathbb{C}_3'}{\mathbb{C}_1[\,a\_!\{b\}\,.\,P\,] \,\big|\, \mathbb{C}_2\lfloor\, \mathbb{C}_3[\,a\hat{}?\{c\}\,.\,Q\,]\,\rfloor \longrightarrow \mathbb{C}_1'[\,P\,] \,\big|\, \mathbb{C}_2'\lfloor\, \mathbb{C}_3'[\,Q[m/p]\,]\,\rfloor}$$

$$\frac{\mathbb{C}_1 \longrightarrow \mathbb{C}_1' \quad \mathbb{C}_2 \longrightarrow \mathbb{C}_2' \quad \mathbb{C}_3 \longrightarrow \mathbb{C}_3'}{\mathbb{C}_1\lfloor\, \mathbb{C}_2[\,a\hat{}!\{b\}\,.\,P\,]\,\rfloor \,\big|\, \mathbb{C}_3[\,a\_?\{c\}\,.\,Q\,] \longrightarrow \mathbb{C}_1'\lfloor\, \mathbb{C}_2'[\,P\,]\,\rfloor \,\big|\, \mathbb{C}_3'[\,Q[m/p]\,]}$$

$$\frac{\mathbb{C}_1 \longrightarrow \mathbb{C}_1' \quad \mathbb{C}_2 \longrightarrow \mathbb{C}_2' \quad \mathbb{C}_3 \longrightarrow \mathbb{C}_3' \quad \mathbb{C}_4 \longrightarrow \mathbb{C}_4'}{\mathbb{C}_1\lfloor\, \mathbb{C}_2[\,a\#!\{b\}\,.\,P\,]\,\rfloor \,\big|\, \mathbb{C}_3\lfloor\, \mathbb{C}_4[\,a\#?\{c\}\,.\,Q\,]\,\rfloor \longrightarrow \mathbb{C}_1'\lfloor\, \mathbb{C}_2'[\,P\,]\,\rfloor \,\big|\, \mathbb{C}_3'\lfloor\, \mathbb{C}_4'[\,Q[m/p]\,]\,\rfloor}$$

**Execution in context:**

$$\frac{\mathbb{C} \longrightarrow \mathbb{C}' \quad P \longrightarrow Q}{\mathbb{C}[\,P\,] \longrightarrow \mathbb{C}'[\,Q\,]} \qquad \frac{P \longrightarrow Q}{(\nu a)\,P \longrightarrow (\nu a)\,Q} \qquad \frac{P \longrightarrow Q}{\lfloor\, P\,\rfloor \longrightarrow \lfloor\, Q\,\rfloor}$$

**Structural congruence:**

$$\frac{P \equiv Q \quad Q \longrightarrow Q' \quad Q' \equiv P'}{P \longrightarrow P'}$$

Fig. 14. Reaction relation of BioAmbients.

semantics, to be investigated in a (presumably) nicer labelled setting. The involved notion of context is not related to ours and calculi with choice are not considered at all.

Larsen [8] uses contexts equipped with structural operational semantics to define a notion of context dependent equivalence. Larsen and Xinxin [9] extends this into a notion of compositionality that allows Hennesy-Milner properties of composite systems to be decomposed into joint properties of the sub-components. This use of active contexts has subsequently been adopted back into the realm of functional languages by Sands [19]. In all cases the contexts are, in some sense, active, but the associated semantics is defined using exactly the complicated label languages that reaction semantics strive to avoid and, in purpose, the approach is unrelated to ours.

## 6 Conclusion

We have developed the notion of *active evaluation contexts* that allows reaction semantics in the style of the Chemical Abstract Machine [2] to be defined for a larger class of process algebras than has previously been considered.

In line with previous work on reaction semantics for CCS [13] we have compared our approach to the more classical approach of structural oper-

ational semantics [11] and proved that the two types of semantics coincide when closed process expressions are considered. This result indicates that the notion of active evaluation contexts constitutes a *sound* approach to reaction semantics.

In order to illustrate our approach on more expressive calculi, such as those that arise to meet the demands of domain specific modelling for complex domains, we have presented a full reaction semantics for an extension of Regev and Cardelli's comprehensive BioAmbients calculus [18] that includes unrestricted choice. The resulting semantics has two properties that we find very encouraging. Firstly the process of actually defining it was highly systematic and, thus, easy. Secondly we find that it is comparable in elegance to Regev's original semantics. This indicates that the notion of active evaluation contexts also constitutes a *sensible* approach to reaction semantics.

Thus, we believe that active evaluation contexts constitute a sound and sensible approach to defining reaction semantics in general. We can only fully substantiate this claim, however, by subjecting other advanced calculi, which combine various features in new ways, to the approach.

## References

[1] Bergstra, J. A. and J. W. Klop, *Algebra of communicating processes*, in: *Proc. of CWI Symposium*, Mathematics and Computer Science (1986), pp. 89–138.

[2] Berry, G. and G. Boudol, *The chemical abstract machine*, in: *POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (1990), pp. 81–94.

[3] Cardelli, L., *Bioware languages*, in: A. Herbert and K. S. Jones, editors, *Computer Systems: Theory, Technology, and Applications - A Tribute to Roger Needham*, Monographs in Computer Science, Springer, 2004 pp. 59–65.

[4] Cardelli, L. and A. D. Gordon, *Mobile Ambients*, Theoretical Computer Science **240** (2000), pp. 177–213.

[5] Felleisen, M. and D. P. Friedman, *Control operators, the SECD-machine, and the λ-calculus*, in: M. Wirsing, editor, *Formal Descriptions of Programming Concepts III*, North-Holland, 1986 pp. 193–219.

[6] Godskesen, J. C., T. Hildebrandt and V. Sassone, *A calculus of mobile resources*, in: L. Brim, P. Jancar, M. ir Kretinsky and A. in Kucera, editors, *CONCUR 13 – Concurrency Theory*, LNCS **2421** (2002), pp. 272–287.
URL citeseer.ist.psu.edu/article/godskesen02calculus.html

[7] Hoare, C. A. R., *Communicating sequential processes*, Commun. ACM **21** (1978), pp. 666–677.

[8] Larsen, K. G., *A context dependent equivalence between processes*, Theor. Comput. Sci. **49** (1987), pp. 185–215.

[9] Larsen, K. G. and L. Xinxin, *Compositionality through an operational semantics of contexts*, Journal of Logic and Computation **1** (1991), pp. 761–795.

[10] Levi, F. and D. Sangiorgi, *Controlling interference in ambients*, in: *Proc. of POPL'2000* (2000), pp. 352–364.

[11] Milner, R., "A Calculus of Communicating Systems," Lecture Notes in Computer Science Vol. 92, Springer-Verlag, 1980.

[12] Milner, R., "Communication and Concurrency," Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[13] Milner, R., "Communicating and Mobile Systems: The $\pi$-Calculus," Cambridge University Press, 1999.

[14] Nielson, H. R., F. Nielson and M. Buchholtz, *Security for mobility*, in: *FOSAD 2001/2002 Tutorial Lecture Notes*, LNCS **2946**, Springer, 2004 pp. 207–265.

[15] Plotkin, G. D., *A Structural Approach to Operational Semantics*, Technical Report DAIMI FN-19, University of Aarhus (1981).
URL citeseer.csail.mit.edu/plotkin81structural.html

[16] Plotkin, G. D., *Personal communication to flemming nielson*, Aarhus (1981).

[17] Regev, A., "Computational Systems Biology: A Calculus for Biomolecular Knowledge," Ph.D. thesis, Tel Aviv University (2003).

[18] Regev, A., E. M, Pamina, W. Silverman, L. Cardelli and E. Shapiro, *BioAmbients: An abstraction for biological compartments*, Theoretical Computer Science **325** (2004), pp. 141–167.

[19] Sands, D., *Towards operational semantics of contexts in functional languages*, in: *Proceedings of the 6th Nordic Workshop on Programming Theory*, 1994, pp. 378–385.
URL citeseer.ist.psu.edu/68354.html

[20] Sewell, P., *From rewrite rules to bisimulation congruences*, Theoretical Computer Science **274** (2002), pp. 183–230.
URL citeseer.ist.psu.edu/sewell98from.html

# Notes on Generative Probabilistic Bisimulation

Simone Tini [1,2]

*Dipartimento di Scienze della Cultura, Politiche e dell'Informazione*
*Università dell'Insubria*
*Via Carloni 78, 22100, Como, Italy*

**Abstract**

In this notes we consider the model of Generative Probabilistic Transition Systems, and Baier and Hermanns' notion of weak bisimulation defined over them. We prove that, if we consider any process algebra giving rise to a Probabilistic Transition System satisfying the condition of regularity and offering prefixing, interleaving, and guarded recursion, then the coarsest congruence that is contained in weak bisimulation is strong bisimulation.

*Key words:* Process Algebras, Generative Probability, Behavioral Equivalences, Bisimulation, Weak Bisimulation, Congruence

## 1 Introduction

*Probabilistic process algebras* have been introduced in the literature (see, among the others, [1,2,7,3,10,11,12,16,18,21,31]) to develop techniques dealing with both functional and non-functional aspects of system behavior, such as performance and reliability. Models of probabilistic processes are classified in [18] into *generative*, *reactive*, and *stratified*. In the generative model, a single probability distribution is ascribed to all moves of a given process, independently of their action label. In the reactive model, the kind of action performed by a given process in chosen in a nondeterministic way, and a probability distribution is ascribed to all moves of that process labeled with that action. In the stratified model a given process has either *probability moves*, to which a single probability distribution is ascribed and that are associated with no action label, or a single *action move*, having an action label, thus implying a clear separation of action and probability. The model of *Probabilistic Automata* [29] was introduced to capture both probability and the classical

[1] Author partially supported by PRIN Project "Analisi di sistemi di Riduzione mediante sistemi di Transizione" (ART)
[2] Email: simone.tini@uninsubria.it

*This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science*

process algebraic notion of nondeterminism. Here, a state of an automaton can have several *transitions* that are chosen in a nondeterministic way, and each transition leads to a probabilistic distribution over action labeled moves. Usually, the model of [29] is known as the *non-alternating model*, in contraposition with the *alternating model* of [19], where there is a clear distinction between *nondeterministic* states, enabling transitions leading to a unique state and that are chosen in a nondeterministic way, and *probabilistic states*, enabling a unique transition leading to a probabilistic distribution over states.

*Probabilistic transition systems* (PTSs, for short), which extend classic labeled transition systems by some mechanism to represent probability, have been employed as a basic semantic model of probabilistic processes. Of course, several definitions of PTS have been introduced, taking into account of the probabilistic model considered. In order to abstract away from irrelevant information on the way that probabilistic processes compute, several notions of probabilistic *equivalence* and *preorder* have been defined over the PTS models [2,8,9,11,12,19,20,22,23,25,30,32]. In order to fit a given equivalence into an axiomatic framework, it is required that it is a *congruence* with respect to all process algebra operations. *Probabilistic bisimulation*, which relates two processes iff their PTSs have the same probabilistic branching structure, and that was originally defined in [25] for the reactive model, enjoys the congruence property in the process algebras proposed in the papers mentioned above, and is one of the equivalence definitions most frequently employed.

In the nonprobabilistic case, *weak bisimulation* has been successfully proposed by Milner [26] as an equivalence relation that abstracts away from *internal* computation steps. A notion of weak bisimulation for the non-alternating model has been considered in [2,11,12,30]. Baier and Hermanns [5] formulated a notion of weak bisimulation inspired by [26] for the generative model. We refer to [5] for interesting motivations and results on probabilistic weak bisimulation.

In the nonprobabilistic setting, it is well known that weak bisimulation is not a congruence with respect to the operation of nondeterministic choice, which is offered by most of known process algebras. Due to the importance of having the congruence property, the coarsest congruence with respect to nondeterministic choice that is finer than weak bisimulation has been characterized, and called *observational congruence* by Milner [26]. Such a congruence is known also with the names of *rooted $\tau$-bisimulation* [4] and *rooted weak bisimulation* [6]. Also in the non-alternating model, the coarsest congruence with respect to nondeterministic choice that is finer than weak bisimulation has been characterized [2,11,12].

Process algebras respecting the generative model do not offer any operation of nondeterministic choice. More precisely, these process algebras do not offer any operation introducing nondeterminism. However, in general, also in the generative model weak bisimulation is not a congruence. In fact, many process algebras offer a parametric version of interleaving operation, where the

parameter determines the probability to move of each of the two composed processes, and we show by means of a very simple counterexample that weak bisimulation is not a congruence with respect to this interleaving operation. Also the CCS-like parallel composition operation of [10,7] and the CSP-like parallel composition operation of [10] do not preserve weak bisimulation.

Our aim is then to study in the generative model the problem to give a characterization of the coarsest equivalence notion being finer than probabilistic weak bisimulation and being a congruence with respect to any reasonable kernel of operations. To this purpose, we assume prefixing, interleaving, and guarded recursion as such a kernel of operations, since they are widely employed. We prove that, if we only consider process algebras giving rise to PTSs satisfying the regularity condition, then the congruence we aim to characterize is probabilistic bisimulation. In some sense, this result has negative consequences. In fact, in the nonprobabilistic case a lot of work has been done on congruences weaker than bisimulation and stronger than weak bisimulation, and in the generative case such a work cannot be repeated, since our result implies that there is no congruence strictly lying between bisimulation and weak bisimulation. Note that our result emphasizes also a difference between the generative and the non-alternating model, where the coarsest congruence contained in weak bisimulation [6,13,14,15,17] is strictly coarsest than strong bisimulation. This difference depends on the fact that the parallel composition operation of the non-alternating model has no parameter, introduces nondeterminism, and can be treated as the classical interleaving operation of [26].

## 2 Probabilistic Bisimulations

Given any set $S$, let $\mathcal{M}(S)$ denote the set of all multisets over $S$. Let us employ "$\{\!|$" and "$|\!\}$" as brackets for multisets.

The following definition originates from [7,3,5].

**Definition 2.1** A *generative probabilistic transition system* (GPTS, for short) is a triple $(\mathcal{S}, Act, T)$, where $\mathcal{S}$ is a set of *states*, $Act$ is a countable set of *actions*, and $T \in \mathcal{M}(\mathcal{S} \times Act \times (0,1] \times \mathcal{S})$ is a multiset of *transitions* such that, for all states $s \in \mathcal{S}$:

$$\sum \{\!|\, p \mid \exists \alpha \in Act, s' \in S : (s, \alpha, p, s') \in T \,|\!\} \in \{0, 1\}$$ [3]

Def. 2.1 requires that each state $s \in \mathcal{S}$ is *semistochastic*, namely, the sum of the probabilities of its outgoing transitions, if there are any, sums up to 1. Let us recall that GPTSs considered in [18,31] have a weaker requirement, since they admit that, for each state $s$, the sum of the probabilities of its

[3] Note that multisets are needed to handle the case where from a state $s$ several transitions with the same label $\alpha$ and probability $p$ lead to a state $s'$.

outgoing transitions, if there are any, is a value $0 \leq q \leq 1$, the interpretation being that $s$ deadlocks with probability $1 - q$. Results proved in the present paper hold also for the model of GPTS of [18,31], since they do not depend on any constraint on the probability of the transitions leaving from $s$.

Let $s \xrightarrow{\alpha,p} s'$ denote that $(s, \alpha, p, s') \in T$, $s \rightarrow$ denote that $s \xrightarrow{\alpha,p} s'$ holds for some $\alpha$, $p$ and $s'$, and $s \not\rightarrow$ denote that $s \xrightarrow{\alpha,p} s'$ holds for no $\alpha$, $p$ and $s'$.

Let $s \Longrightarrow s'$ denote that $s'$ is *reachable* from $s$, namely there exists a sequence of transitions $s_0 \xrightarrow{\alpha_0,p_0} s_1 \ldots s_{n-1} \xrightarrow{\alpha_{n-1},p_{n-1}} s_n$ such that $s_0 = s$ and $s_n = s'$.

In the following we assume the "regularity" condition, namely, for each state $s \in \mathcal{S}$ there are only finitely many outgoing transitions $s \xrightarrow{\alpha,p} s'$, and from $s$ only finitely many other states can be reached through any (possibly infinite) sequence of transitions.

Let us recall the *cumulative probability distribution function* $\mu_G$ [18], which computes the total probability by which from a state $s$ a state $s'$ can be reached through transitions labeled with an action $\alpha$. Adopting the convention that the empty sum of probability is 0, $\mu_G$ is defined as follows.

**Definition 2.2** $\mu_G : \mathcal{S} \times Act \times \mathcal{S} \rightarrow [0, 1]$ is the function given by: $\forall s \in \mathcal{S}$, $\forall \alpha \in Act$, $\forall s' \in \mathcal{S}$:

$$\mu_G(s, \alpha, s') = \sum \{\!|\, p \mid s \xrightarrow{\alpha,p} s' \in T \,|\!\}$$

Function $\mu_G$ can be extended to sets of target states. $\forall s \in \mathcal{S}$, $\forall \alpha \in Act$, $\forall S \subseteq \mathcal{S}$:

$$\mu_G(s, \alpha, S) = \sum_{s' \in S} \mu_G(s, \alpha, s')$$

Following [5], function $\mu_G$ can be extended to sequences of actions in $Act^*$. Let $\epsilon$ denote the empty sequence of actions. For each $\alpha \in Act$ and $\lambda \in Act^*$, let $\alpha\lambda$ denote the sequence in $Act^*$ obtained by prefixing $\lambda$ with $\alpha$.

Then, $\forall s \in \mathcal{S}$, $\forall \alpha \in Act$, $\forall \lambda \in Act^*$, $\forall S \subseteq \mathcal{S}$:

$$\mu_G(s, \epsilon, S) = 1 \text{ if } s \in S$$
$$\mu_G(s, \epsilon, S) = 0 \text{ if } s \notin S$$
$$\mu_G(s, \alpha\lambda, S) = \sum_{s' \in \mathcal{S}} \mu_G(s, \alpha, s') \cdot \mu_G(s', \lambda, S)$$

Finally, following [5] function $\mu_G$ can be extended to sets of sequences of actions in $Act^*$. Let $\Lambda$ denote any subset of $Act^*$, and $\Lambda/\alpha$ denote the set $\{\lambda \in Act^* \mid \alpha\lambda \in \Lambda\}$.

Then, $\forall s \in \mathcal{S}$, $\forall \Lambda \subseteq Act^*$, $\forall S \subseteq \mathcal{S}$:

$$\mu_G(s, \Lambda, S) = 1 \text{ if } \epsilon \in \Lambda \text{ and } s \in S$$
$$\mu_G(s, \Lambda, S) = \sum_{(\alpha, s') \in Act \times \mathcal{S}} \mu_G(s, \alpha, s') \cdot \mu_G(s', \Lambda/\alpha, S) \text{ otherwise}$$

We can recall now the notion of bisimulation [18] for GPTSs. For any

equivalence relation $\mathcal{R}$ over the set of states $\mathcal{S}$, let $\mathcal{S}/\mathcal{R}$ denote the set of equivalence classes induced by $\mathcal{R}$.

**Definition 2.3** An equivalence relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is a *(strong) bisimulation* if $(s_1, s_2) \in \mathcal{R}$ implies: $\forall C \in \mathcal{S}/\mathcal{R}$, $\forall \alpha \in Act$:

$$\mu_G(s_1, \alpha, C) = \mu_G(s_2, \alpha, C)$$

The union of all bisimulations is, in turn, a bisimulation, denoted by $\sim$. Relation $\sim$ equates states having the same probabilistic branching structure.

Let us assume that $Act$ contains the special *silent* action $\tau$. We can recall now Baier and Hermanns' notion of weak bisimulation [5] for GPTSs. Let us denote sets of sequences of actions in $Act^*$ with regular expressions.

**Definition 2.4** An equivalence relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is a *weak bisimulation* if $(s_1, s_2) \in \mathcal{R}$ implies: $\forall C \in \mathcal{S}/\mathcal{R}$, $\forall a \in Act \setminus \{\tau\}$:

$$\mu_G(s_1, \tau^* a \tau^*, C) = \mu_G(s_2, \tau^* a \tau^*, C)$$

$$\mu_G(s_1, \tau^*, C) = \mu_G(s_2, \tau^*, C)$$

The union of all weak bisimulations is, in turn, a weak bisimulation, denoted by $\approx$. Relation $\approx$ is coarser than $\sim$, since it abstracts from silent computation steps.

## 3 Weak Bisimulation is not a Congruence

As usual, let us assume a process description language whose abstract syntax is given by a *signature* $\Sigma$, consisting of a set of *operation symbols* together with an *arity* mapping that assigns a natural $ar(f)$ to every $f \in \Sigma$.

In this setting, a process is a *closed term* over $\Sigma$, where, for a set of *variables* `Var` ranged over by $x, y, \ldots$, the set of (*open*) *terms* over $\Sigma$ and `Var` is the least set such that:

- each variable $x \in$ `Var` is a term;
- $f(t_1, \ldots, t_{ar(f)})$ is a term whenever $f \in \Sigma$ and $t_1, \ldots, t_{ar(f)}$ are terms,

and the terms that do not contain variables in `Var` are called closed terms.

The semantics of the language is given by a GPTS, whose states are processes, and whose transitions are inferred by a set of SOS rules [27,28]. As usual, let us assume that $\Sigma$ contains the operation symbol 0 (sometimes denoted **nil**) with $ar(0) = 0$, where 0 represents the idling process having no move.

Let us recall the notion of congruence.

**Definition 3.1** An equivalence relation $R$ over processes is called a *congruence* iff, for each $f \in \Sigma$, if relation $(t_i, t_i') \in R$ holds for all $1 \leq i \leq ar(f)$, then $(f(t_1, \ldots, t_{ar(f)}), f(t_1', \ldots, t_{ar(f)}')) \in R$.

$$\frac{}{\alpha \cdot x \xrightarrow{\alpha,1} x} \qquad \frac{x[\mathbf{rec}X.\, x/X] \xrightarrow{\alpha,p} y}{\mathbf{rec}X.\, x \xrightarrow{\alpha,p} y}$$

$$\frac{x_1 \xrightarrow{\alpha_1,p_1} y_1 \quad x_2 \not\rightarrow}{x_1 \parallel^p x_2 \xrightarrow{\alpha_1,p_1} y_1 \parallel^p x_2} \qquad \frac{x_1 \not\rightarrow \quad x_2 \xrightarrow{\alpha_2,p_2} y_2}{x_1 \parallel^p x_2 \xrightarrow{\alpha_2,p_2} x_1 \parallel^p y_2}$$

$$\frac{x_1 \xrightarrow{\alpha_1,p_1} y_1 \quad x_2 \rightarrow}{x_1 \parallel^p x_2 \xrightarrow{\alpha_1,p_1 \cdot p} y_1 \parallel^p x_2} \qquad \frac{x_1 \rightarrow \quad x_2 \xrightarrow{\alpha_2,p_2} y_2}{x_1 \parallel^p x_2 \xrightarrow{\alpha_2,p_2 \cdot (1-p)} x_1 \parallel^p y_2}$$

Table 1
Some probabilistic operations; $x, x_1, x_2, y_1, y_2$ are process variables, $\alpha, \alpha_1, \alpha_2$ range over $Act$, $p, p_1, p_2$ are variables over the interval $[0, 1]$, and $X$ is a *recursion variable*.

Now, bisimulation $\sim$ is a congruence with respect to operations of well known process algebras used in the literature [1,7,3,10,16,18,21,24,31].

Let us consider the operation of probabilistic interleaving $\parallel^p$ of [3], whose semantics in SOS style is presented in Table 1. Intuitively, if both processes $t_1$ and $t_2$ can move, then the process $t_1 \parallel^p t_2$ moves as $t_1$ with probability $p$ and as $t_2$ with probability $1 - p$. If only $t_1$ (resp. $t_2$) can move, then $t_1 \parallel^p t_2$ moves as $t_1$ (resp. $t_2$) with probability 1.

By an example, we can show that weak bisimulation is not a congruence with respect to operation $\parallel^p$.

**Example 3.2** Let $a \in Act \setminus \{\tau\}$, and $t_1$ and $t_2$ be the processes $t_1 \equiv \tau \cdot a \cdot 0$ and $t_2 \equiv \tau \cdot \tau \cdot a \cdot 0$, where $\cdot$ is the prefixing operation described in Table 1. It is immediate that $t_1 \approx t_2$, but, for each $0 < p < 1$ and $b \in Act \setminus \{a, \tau\}$, $t_1 \parallel^p b \cdot 0 \not\approx t_2 \parallel^p b \cdot 0$. In fact, $\mu_G(t_1 \parallel^p b \cdot 0, \tau^* a \tau^*, 0 \parallel^p b \cdot 0) = p^2$, whereas $\mu_G(t_2 \parallel^p b \cdot 0, \tau^* a \tau^*, 0 \parallel^p b \cdot 0) = p^3$, and no other state weak bisimilar to $0 \parallel^p b \cdot 0$ is reachable from $t_1 \parallel^p b \cdot 0$ and $t_2 \parallel^p b \cdot 0$. Intuitively, to perform the $a$ move before the $b$ move by $b \cdot 0$, $t_1$ has to win two competitions versus $b \cdot 0$, whereas $t_2$ has to win three competitions.

Note that Example 3.2 holds also if we replace the interleaving operation $\parallel^p$ with the CCS-like parallel composition operation $\parallel^p_q$ of [10,7], provided that $b \neq \overline{a}$, or with the CSP-like parallel composition operation $\parallel^p_A$ of [10], provided that the set of actions $A$ contains neither $a$ nor $b$.

## 4 Bisimulation is the Coarsest Congruence Contained in Weak Bisimulation

Let **rec** be the recursion operation defined in Table 1. We assume that the recursion variables always appear as *guarded*, according to the usual definition. In this section we prove that, if we consider any process algebra offering the operations of prefixing, recursion, and interleaving as in Table 1, then the

coarsest congruence contained in weak bisimulation is bisimulation.

Let us introduce the notion of $p$-bisimulation. It can be viewed as a relation weaker than a bisimulation, in the sense that the probabilistic branching structure of processes is considered modulo the probability value $p$.

**Definition 4.1** Given any $0 \leq p \leq 1$, an equivalence relation $\mathcal{R}_p \subseteq \mathcal{S} \times \mathcal{S}$ is a *$p$-bisimulation* if $(s_1, s_2) \in \mathcal{R}_p$ implies: $\forall C \in \mathcal{S}/\mathcal{R}_p$, $\forall \alpha \in Act$:

$$|\mu_G(s_1, \alpha, C) - \mu_G(s_2, \alpha, C)| \leq p$$

In general, it is not guaranteed that the union of two $p$-bisimulations is a $p$-bisimulation. Hence, $p$-bisimulations are less elegant than bisimulations and weak bisimulations. However, we are not interested here in studying their theory, we simply use $p$-bisimulations in our proofs.

The following result is immediate.

**Proposition 4.2** *A $p$-bisimulation is also a $q$-bisimulation, for each $q > p$. A 0-bisimulation is a bisimulation.*

Given a process $t$, let $actions(t)$ denote the set of the actions appearing in the transition labels in the portion of GPTS rooted in $t$. The regularity condition over the GPTS ensures that $actions(t)$ is a finite set.

Let us assume two arbitrary processes $t$ and $t'$. Since $actions(t)$ and $actions(t')$ are finite sets, and since $Act$ is a countable set, we can assume also two actions $b, c \in Act \backslash (actions(t) \cup actions(t') \cup \{\tau\})$. We can prove that, given arbitrary values $0 < p, q < 1$, then, if relation $(t \parallel^q \mathbf{rec}X\,.\,c{\cdot}X) \parallel^p \mathbf{rec}X\,.\,b{\cdot}X \approx (t' \parallel^q \mathbf{rec}X\,.\,c \cdot X) \parallel^p \mathbf{rec}X\,.\,b \cdot X$ holds, then there exists some $(p \cdot q)$-bisimulation relating $t$ and $t'$.

**Lemma 4.3** *Given arbitrary processes $t, t'$, arbitrary values $0 < p, q < 1$, and any pair of actions $b, c \in Act \setminus (\{\tau\} \cup actions(t) \cup actions(t'))$, it holds that:*

$$(t \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X \approx (t' \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X$$

*implies*

$$t \sim_{p \cdot q} t' \text{ for some } (p \cdot q)\text{-bisimulation } \sim_{p \cdot q}$$

**Proof.**

First of all let us prove the following lemma.

**Lemma 4.4** *For each pair of processes $s \approx s'$ such that $s$ is reachable from $(t \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X$ and $s'$ is reachable from $(t' \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X$, for each action $\alpha \in Act$, and for each equivalence class $C \in \mathcal{S}/\approx$, it holds that $|\mu_G(s, \alpha, C) - \mu_G(s', \alpha, C)| \leq p^2 \cdot q^2$.*

**Proof.** Let us note that $s$ has the form $(t_1 \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X$ and $s'$ has the form $(t_2 \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X$, for some $t_1$ and $t_2$. We can distinguish four cases:

(i) $\alpha = a$, with $a \in Act \setminus \{b, c, \tau\}$.

Let $p_1$ and $p_2$ be the values such that $p_1 = \mu_G(s, a, C)$ and $p_2 = \mu_G(s', a, C)$. We have to prove that $|p_1 - p_2| \leq p^2 \cdot q^2$. The thesis is immediate if $p_1 = p_2$. If $p_1 \neq p_2$, then w.l.o.g. we can assume that $p_1 > p_2$. Now, $p_1 = \mu_G(s, a, C)$ implies $\mu_G(s, \tau^* a \tau^*, C) \geq p_1$. Since $s \approx s'$ we infer that $\mu_G(s', \tau^* a \tau^*, C) \geq p_1$. Since $p_1 > p_2 = \mu_G(s', a, C)$, we infer that $\mu_G(s', \tau^+ a \tau^*, C) + \mu_G(s', a\tau^+, C) \geq (p_1 - p_2)$, where $\tau^+$ denotes the set of all the sequences of $n \geq 1$ $\tau$-actions. Since $a \neq b$ and $a \neq c$, processes in $C$ can be reached from $s'$ by sequences $\tau^+ a \tau^*$ and $a\tau^+$ only through at least two moves by $t_2$. This implies that $\mu_G(s', \tau^+ a \tau^*, C) + \mu_G(s', a\tau^+, C) \leq p^2 \cdot q^2$. Summarizing, $p^2 \cdot q^2 \geq p_1 - p_2$, and the thesis is proved.

(ii) $\alpha = b$.

Since $b \notin actions(t) \cup actions(t')$, the only $b$ move by $s$ is due to $\mathbf{rec}\, X.\, b \cdot X$ and leads to $s$ itself, and, analogously, the only $b$ move by $s'$ is due to $\mathbf{rec}\, X.\, b \cdot X$ and leads to $s'$ itself. Hence, either $C$ contains neither $s$ nor $s'$, and the thesis is immediate since $\mu_G(s, b, C) = 0 = \mu_G(s', b, C)$, or $C$ contains both $s$ and $s'$. Let us concentrate on the second case. Since $t_1 \parallel^q \mathbf{rec} X.\, c \cdot X$ moves, it cannot happen that the probability of the $b$-move by $\mathbf{rec}\, X.\, b \cdot X$ is 1, and we are sure that this $b$ move has probability $1 - p$, and, therefore, $\mu_G(s, b, C) = 1 - p$. For the same reason we infer that $\mu_G(s', b, C) = 1 - p$. Summarizing, $\mu_G(s, b, C) = \mu_G(s', b, C)$, and the thesis is proved.

(iii) $\alpha = c$.

Let $p_1$ and $p_2$ be the values such that $p_1 = \mu_G(s, c, C)$ and $p_2 = \mu_G(s', c, C)$. Since $c \notin actions(t) \cup actions(t')$, the only $c$ move by $s$ is due to $\mathbf{rec}\, X.\, c \cdot X$ and leads to $s$ itself, and, analogously, the only $c$ move by $s'$ is due to $\mathbf{rec}\, X.\, c \cdot X$ and leads to $s'$ itself. Hence, either $C$ contains neither $s$ nor $s'$, and the thesis is immediate since $\mu_G(s, c, C) = 0 = \mu_G(s', c, C)$, or $C$ contains both $s$ and $s'$. Let us concentrate on the second case. It holds that either $p_1 = p \cdot (1 - q)$, if $t_1$ has some move, or $p_1 = p$, if $t_1$ has no move. Moreover, either $p_2 = p \cdot (1 - q)$, if $t_2$ has some move, or $p_2 = p$, if $t_2$ has no move. Therefore, it suffices to prove that it cannot happen that $t_1$ moves and $t_2$ does not to infer that either $p_1 = p \cdot (1 - q) = p_2$ or $p_1 = p = p_2$, which implies the thesis.

By contradiction, let us assume that $t_1$ moves and $t_2$ does not. Since $t_2$ does not move, $s'$ has only one $b$ move with probability $1 - p$ and only one $c$ move with probability $p$. Since $s \approx s'$, we infer that $s$ has only moves in $\{b, c, \tau\}$, thus implying that $t_1$ has $\tau$ moves. This implies that the overall probability of sequences $\tau^* b \tau^*$ by $s$ is strictly greater than $1 - p$, which contradicts that $s \approx s'$.

(iv) $\alpha = \tau$.

First of all let us note that, since $s \approx s'$, either $C$ contains both $s$ and

$s'$, or $C$ contains neither $s$ nor $s'$. If $C$ contains neither $s$ nor $s'$, we can reason as in case (i).

Let us assume that $C$ contains both $s$ and $s'$. In this case, let $p_1$ and $p_2$ be the values such that $p_1 = \mu_G(s, \tau, C)$ and $p_2 = \mu_G(s', \tau, C)$. We have to prove that $|p_1 - p_2| \leq p^2 \cdot q^2$. The thesis is immediate if $p_1 = p_2$. If $p_1 \neq p_2$, then w.l.o.g. we can assume that $p_1 > p_2$. First of all let us note that, since each state $\hat{s}$ reachable from $s$ or $s'$ can perform the $c$ move, it cannot happen that $\hat{s} \xrightarrow{b,1} \hat{s}$, and, therefore, we are sure that $\hat{s} \xrightarrow{b,1-p} \hat{s}$. Since both $s$ and the processes in $C$ reachable through one $\tau$ move from $s$ (with total probability $p_1$) can perform $b$ with probability $1-p$ while remaining in $C$, it holds that $\mu_G(s, \tau^* b \tau^*, C) \geq (1-p) + p_1 \cdot (1-p)$. Since $s \approx s'$, it holds that $\mu_G(s', \tau^* b \tau^*, C) \geq (1-p) + p_1 \cdot (1-p)$. Since states in $C$ cannot perform $b$ with probability 1, we know that $\mu_G(s', b, C) = (1-p)$ and $\sum_{m+n=1} \mu_G(s', \tau^m b \tau^n, C) = p_2 \cdot (1-p)$. Hence, $\sum_{m+n\geq 2} \mu_G(s', \tau^m b \tau^n, C) \geq (p_1 - p_2)(1-p)$. Moreover, we know that $\sum_{m+n\geq 2} \mu_G(s', \tau^m b \tau^n, C) \leq p^2 \cdot q^2 \cdot (1-p)$, since $\tau^m b \tau^n$ with $m+n \geq 2$ requires at least two moves by $t_2$ and one move from $\mathbf{rec}\, X.\, b \cdot X$. Summarizing, $p^2 \cdot q^2 \cdot (1-p) \geq (p_1 - p_2) \cdot (1-p)$, which implies $p^2 \cdot q^2 \geq p_1 - p_2$, and the thesis is proved.

□

Lemma 4.4 implies that there is a $(p^2 \cdot q^2)$-bisimulation relating processes reachable from $(t \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X$ and $(t' \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X$ and that contains the pair formed by $(t \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X$ and $(t' \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X$. Let $\sim_{p^2 \cdot q^2}$ denote such a $(p^2 \cdot q^2)$-bisimulation.

Let us take any equivalence class $C \in \mathcal{S}/\sim_{p^2 \cdot q^2}$. We prove below that the set of processes $\hat{C} = \{s \text{ such that } (s \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X \in C\}$ are an equivalence class of a $(p \cdot q)$-bisimulation relating processes reachable from $t$ and $t'$. Let $\sim_{p \cdot q}$ denote such a $(p \cdot q)$-bisimulation. Relations $\sim_{p \cdot q}$ equates $t$ and $t'$, since $(t \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X$ and $(t' \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X$ are equated by $\sim_{p^2 \cdot q^2}$. The thesis follows from $t \sim_{p \cdot q} t'$.

Hence, it remains to prove that $\hat{C} \in \mathcal{S}/\sim_{p \cdot q}$ for some $(p \cdot q)$-bisimulation $\sim_{p \cdot q}$. Given arbitrary processes $t_1, t_2 \in \hat{C}$, any equivalence class $\hat{D}$, and any action $\alpha \in actions(t_1) \cup actions(t_2)$, the semantics of $\parallel^p$ and $\parallel^q$ implies that:
$\mu_G(t_1, \alpha, \hat{D}) = \frac{1}{p \cdot q} \cdot \mu_G((t_1 \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X, \alpha, D)$,
$\mu_G(t_2, \alpha, \hat{D}) = \frac{1}{p \cdot q} \cdot \mu_G((t_2 \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^q \mathbf{rec}\, X.\, b \cdot X, \alpha, D)$.
Since $(t_1 \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X \sim_{p^2 \cdot q^2} (t_2 \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X$, we are sure that $|\mu_G((t_1 \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X, \alpha, D) - \mu_G((t_2 \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X), \alpha, D)| \leq p^2 \cdot q^2$. Therefore, $|\mu_G(t_1, \alpha, \hat{D}) - \mu_G(t_2, \alpha, \hat{D})| \leq p \cdot q$, as required. □

At first glance, our choice of using a context with two occurrences of the interleaving operator in Lemma 4.3 could be surprising. One could expect that

a context with only one of these occurrence suffices. The point is that the proof of Lemma 4.4 does not work if we consider the context $\_ \parallel^p \mathbf{rec}X\,.\,b{\cdot}X$ instead of $(\_ \parallel^q \mathbf{rec}\,X.\,c\cdot X) \parallel^p \mathbf{rec}\,X.\,b\cdot X$. In fact, both in the proof of case $\alpha = b$, and in the proof of the case $\alpha = \tau$, we exploit the fact that the probability of the $b$ move by $\mathbf{rec}\,X.\,b\cdot X$ cannot be 1. This is implied by the fact that $\mathbf{rec}\,X.\,c\cdot X$ can perform the $c$ move. Without process $\mathbf{rec}\,X.\,c\cdot X$, the process on the left side of $\parallel^p$ could be a process without any move, and the $b$ move by $\mathbf{rec}\,X.\,b\cdot X$ could have probability 1.

Notice that Lemma 4.3 above holds also if we replace the interleaving operations $\parallel^p$ and $\parallel^q$ with the CCS-like parallel composition operations $\parallel^p_{p'}$ and $\parallel^q_{q'}$ of [10,7], provided that $t$ and $t'$, besides performing neither $b$ nor $c$, performs neither $\overline{b}$ nor $\overline{c}$. Moreover, Lemma 4.3 holds also if we replace $\parallel^p$ and $\parallel^q$ with the CSP-like parallel composition operations $\parallel^p_A$ and $\parallel^q_A$ of [10], with $A = \emptyset$.

Let us prove now that processes related by $p$-bisimulations for all $0 < p < 1$ are strong bisimilar.

**Lemma 4.5** *Given processes $t$ and $t'$, if for each $0 < p < 1$ there exists a $p$-bisimulation $\sim_p$ such that $t \sim_p t'$, then it holds that $t \sim t'$.*

**Proof.** Since the regularity condition ensures that the number of states reachable from $t$ and $t'$ is finite, there exists an equivalence relation $\mathcal{R}$ over the states reachable from $t$ and $t'$ such that $(t, t') \in \mathcal{R}$ and such that, given any $\delta > 0$, $\mathcal{R}$ is an $\epsilon$-bisimulation for infinite many $\delta > \epsilon > 0$. We can prove that $\mathcal{R}$ is a strong bisimulation. By contradiction, let us assume that $\mathcal{R}$ is not a strong bisimulation. Then, there exists a pair of states $(s, s') \in \mathcal{R}$, an action $\alpha \in Act$, and an equivalence class $C$ over $\mathcal{R}$ such that $\mu_G(s, \alpha, C) \neq \mu_G(s', \alpha, C)$. Let $d$ be the value $|\mu_G(s, \alpha, C) - \mu_G(s', \alpha, C)|$. It follows that $\mathcal{R}$ is not an $\epsilon$-bisimulation for any $\epsilon < d$, which contradicts that $\mathcal{R}$ is an $\epsilon$-bisimulation for infinite many $d > \epsilon > 0$. Now, since $\mathcal{R}$ is a bisimulation and $(t, t') \in \mathcal{R}$, the thesis holds. □

We can give now our main result.

**Theorem 4.6** *Given arbitrary processes $t$ and $t'$, if for all $0 < p, q < 1$ and actions $b, c \in Act \setminus (\{\tau\} \cup actions(t) \cup actions(t'))$ it holds that $(t \parallel^q \mathbf{rec}\,X.\,c\cdot X) \parallel^p \mathbf{rec}\,X.\,b\cdot X \approx (t' \parallel^q \mathbf{rec}\,X.\,c\cdot X) \parallel^p \mathbf{rec}\,X.\,b\cdot X$ then it follows that $t \sim t'$.*

**Proof.** By Lemma 4.3, for each $0 < p, q < 1$, it holds that $t \sim_{p\cdot q} t'$ for some $(p \cdot q)$-bisimulation $\sim_{p\cdot q}$. Given any $0 < d < 1$, we can choose $p = q = \sqrt{d}$, to infer that $t \sim_d t'$. Since $t \sim_d t'$ for all $0 < d < 1$, we can apply Lemma 4.5 to infer $t \sim t'$, and the proof is complete. □

Let us assume that $\mathcal{R}$ is an equivalence relation over processes being a congruence w.r.t interleaving, prefixing and recursion, and being finer than

weak bisimulation (i.e. $\mathcal{R} \subset \approx$). Given processes $t$ and $t'$ such that $(t, t') \in \mathcal{R}$, since $\mathcal{R}$ is a congruence, we infer that, for all $0 < p, q < 1$, $((t \parallel^q \mathbf{rec}\, X.\, c{\cdot}X) \parallel^p \mathbf{rec}\, X.\, b \cdot X,\ (t' \parallel^q \mathbf{rec}\, X.\, c \cdot X) \parallel^p \mathbf{rec}\, X.\, b \cdot X) \in \mathcal{R}$. Since $\mathcal{R} \subset \approx$, we infer $t \approx t'$ and $(t \parallel^q \mathbf{rec}\, X.\, c{\cdot}X) \parallel^p \mathbf{rec}\, X.\, b{\cdot}X \approx (t' \parallel^q \mathbf{rec}\, X.\, c{\cdot}X) \parallel^p \mathbf{rec}\, X.\, b{\cdot}X$. Thm. 4.6 implies that $t \sim t'$. Hence, $\mathcal{R} \subseteq \sim$. Since $\sim$ is a congruence w.r.t. interleaving, prefixing and recursion, we infer that $\sim$ is the coarsest congruence contained in $\approx$ that is a congruence w.r.t. these operations.

## 5 Conclusions

We have proved that, if one considers process algebras giving rise to GPTSs satisfying the regularity condition and offering recursion, interleaving and prefixing, then strong bisimulation is the coarsest congruence contained in weak bisimulation. This differentiates the generative probabilistic model not only with respect to the nonprobabilistic case, where interesting congruences strictly lying between strong and weak bisimulation have been studied [6,13,14,15,17], but also with respect to the non-alternating model, where the coarsest congruence being finer than weak bisimulation has been characterized and proved to be coarser than strong bisimulation [2,11,12].

Analogies between the nonprobabilistic and the non-alternating model arise since in the non-alternating model process algebras offer parallel composition operations and a nondeterministic choice operation having the same nature of those of nonprobabilistic process algebras. To support this observation the fact that the axiomatization of the coarsest congruence being finer than weak bisimulation requires rules similar to Milner's "expansion law" to manage parallel composition [12] and rules similar to Milner's "$\tau$-law" to manage $\tau$ prefixing [2,11,12].

In the generative model, no operation introducing nondeterminism is allowed. Therefore one cannot hope to treat parallel composition operations as in the nonprobabilistic case. Asynchronous parallel composition operations require parameters specifying the probability to move of each of the processes running in parallel. These parametric operations do not preserve weak bisimulation, since they distinguish $\tau^m \cdot t_1$ and $\tau^n \cdot t_1$, when $m \neq n$. In fact, when we compose in parallel $\tau^m \cdot t_1$ with another process $t_2$, the $\tau$ actions of $\tau^m \cdot t_1$ imply that actions of $t_1$ can be performed only after $\tau^m \cdot t_1$ has won $m$ competitions versus $t_2$ to perform the $m$ occurrences of $\tau$, and each of these competitions is not for free, meaning that the probability of winning it is not 1 but depend on the parameter of the operation. The ability of discriminating $\tau^m \cdot t_1$ and $\tau^n \cdot t_1$ has as a consequence that there is no congruence strictly lying between strong and weak bisimulation.

Checking whether our result holds also for GPTSs that do not satisfy the regularity condition, and for transition systems respecting the reactive and stratified models of probabilistic processes could be interesting developments of the present notes.

## References

[1] A. Aldini, M. Bravetti, and R. Gorrieri: A Process-algebraic Approach for the Analysis of Probabilistic Non-interference. J. Comput. Secur. 12(2), 2004, 191–245.

[2] E. Bandini and R. Segala: Axiomatizations for Probabilistic Bisimulation. Proc. Int. Coll. on Automata, Languages and Programming, Lecture Notes in Computer Science 2076, Springer, Berlin, 2001, 370–381.

[3] J. C. M. Baeten, J. A. Bergstra, and S. A. Smolka: Axiomatizing Probabilistic Processes: ACP with Generative Probabilities. Inf. Comput. 121(2), 1995, 234–255.

[4] J. C. M. Baeten and W. P. Weijland: Process Algebra. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, 1990.

[5] C. Baier and H. Hermanns: Weak Bisimulation for Fully Probabilistic Processes. Proc. Int. Conf. on Computer Aided Verification, Lecture Notes in Computer Science 1254, Springer, Berlin, 1997, 119–130.

[6] B. Bloom: Structural Operational Semantics for Weak Bisimulation. Theor. Comput. Sci. 146(1–2), 1995, 25–68.

[7] M. Bravetti and A. Aldini: Discrete Time Generative-reactive Probabilistic Processes with Different Advancing Speeds. Theor. Comput. Sci. 290(1), 2003, 355–406.

[8] I. Christoff: Testing Equivalences and Fully Abstract Models for Probabilistic Processes. Proc. Int. Conf. on Concurrency Theory, Lecture Notes in Computer Science 458, Springer, Berlin, 1990, 126–140.

[9] R. Cleaveland, S. A. Smolka, and A. Zwarico: Testing Preorders for Probabilistic Processes. Proc. Int. Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 623, Springer, Berlin, 1992, 708–719.

[10] P. R. D'Argenio, H. Hermanns, and J. P. Katoen: On Generative Parallel Composition. Proc. Int. Work. on Probabilistic Methods in Verification, Electr. Notes Theor. Comput. Sci. 22, Elsevier, Amsterdam, 1999.

[11] Y. Deng and C. Palamidessi: Axiomatizations for Probabilistic Finite-State Behaviors. Proc. Int. Conf. on Foundations of Software Science and Computational Structures, Lecture Notes in Computer Science 3441, Springer, Berlin, 2005, 110–124.

[12] Y. Deng, C. Palamidessi, and J. Pang: Compositional Reasoning for Probabilistic Finite-State Behaviors. In Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday, Lecture Notes in Computer Science 3838, Springer, Berlin, 2005, 309–337.

[13] W. J. Fokkink: Rooted Branching Bisimulation as a Congruence. J. Comput. Syst. Sci. 60(1), 2000, 13–37.

[14] W. J. Fokkink, R. J. van Glabbeek, and P. de Wind: Divide and Congruence Applied to Eta-bisimulation. Proc. Workshop on Structural Operational Semantics, Electr. Notes Theor. Comput. Sci. 156(1), Elsevier, Amsterdam, 2005, 97–113.

[15] W. J. Fokkink, R. J. van Glabbeek, and P. de Wind: Divide and Congruence: From Decomposition of Modalities to Preservation of Branching Bisimulation. Proc. Int. Symp. on Formal Methods for Components and Objects, Lecture Notes in Computer Science 4111, Springer, Berlin, 2005.

[16] A. Giacalone, C. C. Jou, and S. A. Smolka: Algebraic Reasoning for Probabilistic Concurrent Systems. Proc. IFIP Work. Conf. on Programming, Concepts and Methods, 1990, 443–458.

[17] R. J. van Glabbeek: On Cool Congruence Formats for Weak Bisimulations. Proc. Int. Colloquium on Theoretical Aspects of Computing, Lecture Notes in Computer Science 3722, Springer, Berlin, 2005, 331–346.

[18] R. J. van Glabbeek, S. A. Smolka, and B. Steffen: Reactive, Generative and Stratified Models of Probabilistic Processes. Inf. Comput. 121(1), 1995, 59–80.

[19] H. Hansson and B. Jonsson: A Framework for Reasoning about Time and Reliability. Proc. IEEE Real-Time Systems Symposium, IEEE Press, 1989, 102-111.

[20] B. Jonsson and K. G. Larsen: Specification and Refinement of Probabilistic Processes. Proc. IEEE Symp. on Logic in Computer Science, IEEE Press, 1991, 266–277.

[21] B. Jonsson, K. G. Larsen, and W. Yi: Probabilistic Extensions of Process Algebras. Handbook of Process Algebra, Elsevier, Amsterdam, 2001.

[22] B. Jonsson and W. Yi: Compositional Testing Preorders for Probabilistic Processes. Proc. IEEE Symp. on Logic in Computer Science, IEEE Press, 1995, 431–443.

[23] C. C. Jou and S. A. Smolka: Equivalences, Congruences and Complete Axiomatizations for Probabilistic Processes. Proc. Int. Conf. on Concurrency Theory, Lecture Notes in Computer Science 458, Springer, Berlin, 1990, 367–383.

[24] R. Lanotte and S. Tini: Probabilistic Congruence for Semistochastic Generative Processes. Proc. Int. Conf. on Foundations of Software Science and Computational Structures, Lecture Notes in Computer Science 3441, Springer, Berlin, 2005, 63–78.

[25] K. G. Larsen and A. Skou: Bisimulation Trough Probabilistic Testing. Inf. Comput. 94(1), 1991, 1–28.

[26] R. Milner: Communication and Concurrency. Prentice Hall, London, 1989.

[27] G. Plotkin: A Structural Approach to Operational Semantics. Tech. Rep. DAIMI FN-19, University of Aarhus, 1981.

[28] G. Plotkin: A Structural Approach to Operational Semantics. J. Log. Algebr. Program. 60–61, 2004, 17–139.

[29] R. Segala: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD Thesis, MIT, Technical Report MIT/LCS/TR-676, 1995.

[30] R. Segala and N. Lynch: Probabilistic Simulations for Probabilistic Processes. Proc. Int. Conf. on Concurrency Theory, Lecture Notes in Computer Science 836, Springer, Berlin, 1994, 481–496.

[31] E. W. Stark and S. A. Smolka: A Complete Axiom System for Finite-State Probabilistic Processes. In Proof, Language and Interaction: Essays in Honor of Robin Milner, G. Plotkin, C.P. Stirling, and M. Tofte, Eds., MIT Press, 1999.

[32] S. Yuen, R. Cleaveland, Z. Dayar, and S. A. Smolka: Fully Abstract Characterizations of Testing Preorders for Probabilistic Processes. Proc. Int. Conf. on Concurrency Theory, Lecture Notes in Computer Science 863, Springer, Berlin, 1994, 497–512.

# Self-assembling Trees

Vincent Danos [a], Jean Krivine [b], Fabien Tarissan [c]

[a] *Équipe PPS, CNRS & Université Paris VII*

[b] *INRIA Rocquencourt & Université Paris VI*

[c] *Équipe PPS, CNRS & Université Paris VII*

**Abstract**

RCCS is a variant of Milner's CCS where processes are allowed a controlled form of backtracking. It turns out that the RCCS reinterpretation of a CCS process is equivalent, in the sense of weak bisimilarity, to its causal transition system in CCS. This can be used to develop an efficient method for designing distributed algorithms, which we illustrate here by deriving a distributed algorithm for assembling trees. Such a problem requires solving a highly distributed consensus, and a comparison with a traditional CCS-based solution shows that the code we obtain is shorter, easier to understand, and easier to prove correct by hand, or even to verify.

## 1 Introduction

We propose in this paper to illustrate a method for deriving distributed algorithms. The broad idea is to solve a simpler problem, and then reinterpret the obtained solution assuming a generic distributed backtracking mechanism. This is reminiscent of the classic breakdown of solutions to NP problems into an exploration (guessing the solution) and a verification phase (checking the guess is correct). It is also reminiscent of simulated annealing methods where a locally-driven search is backed by a random perturbation. Another analogy is with declarative programming where terse solutions can be obtained because the ambient evaluation framework includes a generic enumeration mechanism.

It turns out that the notion of a solution to a simpler problem can be neatly characterised in terms of the theory of concurrent systems, using the notion of causal transition system, and so does the correctness of the generic backtracking mechanism. A rather general result then ensures that the reinterpreted solution is indeed a solution to the original problem [4].

This compares best with direct approaches when the problem of interest needs reaching a consensus which is itself highly distributed. Thus, for the purpose of illustrating the method, we choose a class of problems which is a simple idealisation of the phenomenon of self-assembly, where simple parts

*This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science*

assemble in some predefined spatial arrangement by means of local and asynchronous interactions. Solutions of such problems indeed involve arbitrarily complex distributed consensus.

Specifically, we derive a distributed algorithm for an ensemble of processes to self-assemble in patterns described as trees. To formulate the algorithm, we use a partially reversible derivative of CCS [12], called RCCS, which introduces a distinction between reversible and irreversible computation steps, together with a notion of distributed memory which allows backtracking reversible steps [3].

The algorithm itself is obtained indirectly. One first defines a simple CCS algorithm such that any allowed tree construction can be simulated, and conversely all trees resulting from a series of local interactions are allowed. This is not yet a solution since the induced assembly may deadlock, but it gets very close to being one. Indeed, by merely reinterpreting the same algorithm in RCCS, and thus allowing backtrack on reversible actions, one obtains a real solution. For the sake of evaluating the method we compare the first algorithm with a direct solution in CCS which explicitly copes with deadlocks. One sees clearly that the latter is both harder to understand, and to prove correct, and also assumes more computational power from the basic processes.

There are limitations to this method. It is likely to provide significantly simpler solutions only to problems in need of complex consensus. Another limitation is that it is for the moment restricted to problems the solution of which can be expressed in CCS. However, recent developements show that correct backtracking mechanisms can be derived for a vastly more comprehensive SOS-based class of agent-languages [15], and that the reinterpretation theorem can be made to bear in the abstract framework of monoidal categories, and thus also covers more general grounds, such as Petri Nets [5].

The paper is self-contained but for the more technical notion of causality which is treated informally; a rigorous treatment is given in ref. [3,4]. Sec. 2 presents the self assembly specification; Sec. 3 introduces the algorithm in CCS; Sec. 4 shows that although it may deadlock, it is well designed in that its causal computations are as in the specification, and that it is therefore correct in RCCS; Sec. 5 compares with a direct solution in CCS. [1]

## 2 Specification

The aim of this section is to define the specification for our distributed implementation as a labelled transition system (LTS).

[1] A preliminary version of this work was presented as a poster at the $7^{th}$ International Conference on Artificial Evolution, Lille, France, Oct 26–28, 2005.

### *2.1 Transition systems and Bisimulation*

A *labelled transition system* consists of a triple: a state space $S$, a set of labels (or actions) $L$, and for each $l \in L$, a binary relation over $S$, written $\rightarrow_l$ and called the *transition relation*. Sometimes one also adds an initial state $s_0 \in S$ to the preceding data. We will write $\rightarrow_w$, with $w = l_1 \cdots l_n$ a word over $L$, for the composite relation $\rightarrow_{l_1}; \cdots ; \rightarrow_{l_n}$.

Given some specification of a distributed system (such as the one given below in this section), and another LTS (possibly obtained from a CCS process as in Sec. 3) believed to be an implementation, one needs some means of stating the correctness of the implementation with respect to the specification. This is given by the notion of *bisimulation*.

Specifically, suppose given two LTSs $(S, s_0, L, \rightarrow)$, $(S', s'_0, L', \rightarrow')$, and a relation $\Phi$ over $L \times L'$. Define the *domain* of $\Phi$ as $\{l \in L \mid \exists l' \in L' : (l, l') \in \Phi\}$, and the *codomain* of $\Phi$ as the domain of the converse relation $\Phi^{-1}$.

Given words $w$, $w'$ over $L$, $L'$: define $w_\Phi$ ($w'_\Phi$) as the word $w$ with all occurrences of labels not in the domain (codomain) of $\Phi$ erased, and write $(w, w') \in \Phi$ if $w_\Phi = l_1 \cdots l_n$ and $w'_\Phi = l'_1 \cdots l'_n$ have the same length, and for all $1 \leq i \leq n$, $(l_i, l'_i) \in \Phi$. Actions in the domain (codomain) of $\Phi$ will be called *visible*, and $\Phi$ itself will be called a *visibility relation*, thus $w_\Phi$ represents the actions in $w$ which are visible according to $\Phi$.

One then says a relation $\simeq$ over $S \times S'$ is a $\Phi$-*bisimulation*, if $s_0 \simeq s'_0$, and whenever $s \simeq s'$:
– if $s \rightarrow_w t$, then $s' \rightarrow'_{w'} t'$, with $(w, w') \in \Phi$ and $t \simeq t'$,
– if $s' \rightarrow'_{w'} t'$, then $s \rightarrow_w t$, with $(w, w') \in \Phi$ and $t \simeq t'$.

The two conditions above are symmetric and state that whatever series of visible actions one LTS may perform, the other may match. In other words the two LTSs, different as they may be, are indistinguishable by synchronisation on visible actions; one says they are $\Phi$-*bisimilar*.

In the context of CCS (see Sec. 3), one has a distinguished silent action, written $\tau$, and setting $L = L'$, and $\Phi = \{(l, l) \mid l \neq \tau\}$ obtains what is known as weak bisimulation. Only non-silent actions, as the name suggests, are observed. An even more stringent case is when $\Phi$ is the identity relation, *i.e.*, all actions are visible, and one speaks of strong bisimulation. Our slight generalisation where the two LTSs use different sets of actions, and some flexibility is allowed regarding which actions are visible and how they match, will be convenient.

### *2.2 The specification*

Let $V$ be a set of nodes given together with a *degree* map $\delta : V \rightarrow \mathbb{N}$ stipulating how many nodes a given node may connect to. The trees considered here will be represented as:

$$t ::= (a, \{t_1, \ldots, t_n\})$$

where $a \in V$ and $n \geq 0$. Hence the simplest tree is $(a, \emptyset)$ which will be simply denoted $a$. Other examples are $(a, \{b, c\})$ where $a$ has two children, $b$ and $c$, and $(a, \{(b, \{c\})\})$ where $a$ and $b$ each have one child, $b$ and $c$. A childless node will be called a leaf as usual. Trees will be considered to be commutative, that is to say for instance $(a, \{b, c\})$ and $(a, \{c, b\})$ stand for the same tree, as the set notation suggests.

A tree $t$ will be said to be *coherent* if all nodes in $t$ have their degree as prescribed by the degree map $\delta$, which means in particular that leaves in $t$ will have arity smaller than 1 (and exactly 1 if they are not also the root of $t$). Imagine for instance that $\delta(a) = 2$, and $\delta(b) = \delta(c) = 1$, then $(a, \{b, c\})$ is coherent, while $(a, \{(b, \{c\})\})$ is not. Also one has that $a$ is coherent if and only if $\delta(a) = 0$. Finally, we will write $n(t)$ to denote the nodes of $t$.

A state of our specification LTS is defined as a pair $(N, \sum_i t_i)$ where $N \subseteq V$ represent the free nodes, and each $t_i$ is a coherent tree representing the trees already built. We write $+$ both for the addition of multisets and the disjoint union of sets. Labels are coherent trees over $V$, and transitions are given as follows:

$$N + n(t), \sum_i t_i \rightarrow_t N, t + \sum_i t_i$$

Note that coherence is the only constraint on trees grown out of our starting set of nodes $V$. Instead, one could choose a different rule for growing trees, by specifying from the outset which trees are allowed. We opt here for the local growth rule, since it allows for simpler notations, and the method given here can anyway be readily adapted to the global growth case.

## 3 Implementation

To define agents showing a collective behaviour in accordance with the specification given above, we use CCS [12], where the only means of communication between agents are binary synchronisations through complementary actions. This restriction translates effectively the intuitive constraint on self-assembly, namely that the global behaviour should be obtained only by means of local interaction.

### *3.1 CCS*

CCS processes have the form:

$$p ::= 0 \mid \sum \alpha_i.p_i \mid (p \mid p) \mid (a)p \mid D(\tilde{x})$$

where $\alpha ::= a \mid \bar{a} \mid \tau$ can be a reception, an emission, or a silent action, and $D(\tilde{x})$ stands for parametric recursive definitions. Sums are taken finite, and the empty sum is denoted by 0 and called the zero process. Structural congruence, written $\equiv$, is the least equivalence relation over processes closed

$$\text{act}\ \frac{}{\sum \alpha_i.p_i \rightarrow_{\alpha_i} p_i} \qquad \text{par}\ \frac{p \rightarrow_\alpha p'}{p \mid q \rightarrow_\alpha p' \mid q} \qquad \frac{p \rightarrow_\alpha p' \quad \alpha \neq a, \bar{a}}{(a)p \rightarrow_\alpha (a)p'}\ \text{res}$$

$$\text{syn}\ \frac{p \rightarrow_\alpha p' \quad q \rightarrow_{\bar{\alpha}} q'}{p \mid q \rightarrow_\tau p' \mid q'}$$

Fig. 1. CCS labelled transition system.

$$\text{NODE}_i =_{def} \tau.(\text{BUILD}_i^{\delta(i)} \mid \text{WAIT}_{i\star}^{\delta(i)}) + \sum_{j \in V} r_{ij}.(\text{BUILD}_i^{\delta(i)-1} \mid \text{WAIT}_{ij}^{\delta(i)-1}) \quad (1)$$

$$\text{BUILD}_i^{n+1} =_{def} \sum_{j \in V} \bar{r}_{ij}.\text{BUILD}_i^n,\ \text{BUILD}_i^0 := 0 \quad (2)$$

$$\text{WAIT}_{i\alpha}^{n+1} =_{def} w_i.\text{WAIT}_{i\alpha}^n,\ \text{WAIT}_{ij}^0 =_{def} \bar{w}_j.\uparrow_j^i,\ \text{WAIT}_{i\star}^0 =_{def} \underline{ok_i}.\uparrow_\star^i \quad (3)$$

Fig. 2. Self-assembly.

under sum, product and restriction, and such that sum and product are associative and commutative and have 0 as neutral element. One also assumes $\alpha$-conversion (renaming), and the following rule to unfold recursive definitions: $D(\tilde{x}) \equiv p$ if $D(\tilde{x}) =_{def} p$ . Thereafter processes are all considered up to $\equiv$.

The CCS labelled transition system given in Fig. 1 explains how a process behaves in terms of the actions it can perform. Thus any CCS process generates an LTS, where states are processes, and labels are CCS actions.

We fix a countable subset $K$ of CCS actions, shown as underlined in the various examples below; these are to be later interpreted as irreversible actions in RCCS, and play no specific role in the CCS semantics.

### *3.2 The implementation*

With both our specification and agent language in place, we turn to the definition of the CCS process describing how agents interact in order to self-assemble. The definition is given in Fig. 2, with $n$ an integer, $i, j \in V$, $\alpha \in V + \{\star\}$, and $\delta$ the degree function described earlier. Each node is translated as a specific agent $\text{NODE}_i$, with $i \in V$. An agent can either decide to be the root of a new tree (left hand side of the choice in (1)), or be recruited by another agent (right hand side of the choice in (1)). In both cases, two subprocesses are spawned, $\text{BUILD}_i^n$, and $\text{WAIT}_{i\alpha}^n$, where $n$ is the number of nodes the agent needs to recruit, as determined by its degree $\delta(i)$; $\alpha$ stands for the agent parent, if any, or for $\star$ if the agent is a root. The process $\text{BUILD}_i^n$ (2) uses $r_{ij}$ to recruit $n$ free agents, while $\text{WAIT}_{i\alpha}^n$ (3) uses $w_j$ to get confirmations of these recruitments, and then uses $\bar{w}_j$ to send a confirmation to its parent. In the special case the agent is the root of the tree, and has no parent, it performs instead the final underlined action $\underline{ok_i}$ to indicate the end of the construction.

There is no intrinsic reason why WAIT should gather confirmations in sequence; this is due to the restrictive syntax of CCS which does not allow prefixing by a set of actions (see for instance ref. [2, Sec. 3]). Likewise, using a richer language such as $\pi$-calculus [13] would make a more elegant code, replacing the $r_{ij}$s with a public name (see ref. [6, Sec. 8]). That would also need a $\pi$-calculus analog of RCCS (see ref. [10, Chap. 9]), and this simple CCS version, perfectible as it is, shall be enough for our illustrative purposes.

One could set the final state of an agent to be simply a zero process, but our convention to take it to be a loop process $\uparrow^i_\alpha =_{def} \tau.\ \uparrow^i_\alpha$, indicating that agent $i$ was successfully recruited by agent $\alpha$, makes it slightly easier to extract the tree a given process has actually finished to build.

The complete system is represented as the product of all agents where all actions but the final $\underline{ok}_i$s are restricted.

### *3.3 Examples*

Here is a computation example with $\delta(a) = 2$, $\delta(b) = \delta(c) = 1$:

$$\begin{aligned}
\text{NODE}_a \mid \text{NODE}_b \mid \text{NODE}_c \longrightarrow\ & \text{BUILD}^2_a \mid \text{WAIT}^2_{a\star} \mid \text{NODE}_b \mid \text{NODE}_c \\
\longrightarrow^\star\ & \text{WAIT}^2_{a\star} \mid \text{WAIT}^0_{ba} \mid \text{WAIT}^0_{ca} \\
\equiv\ & w_a.w_a.\underline{ok_a}.\ \uparrow^a_\star \mid \bar{w}_a.\ \uparrow^b_a \mid \bar{w}_a.\ \uparrow^c_a \\
\longrightarrow^\star\ & \underline{ok_a}.\ \uparrow^a_\star \mid \uparrow^b_a \mid \uparrow^c_a \\
\longrightarrow_{\underline{ok_a}}\ & \uparrow^a_\star \mid \uparrow^b_a \mid \uparrow^c_a
\end{aligned}$$

This corresponds to a single transition $\{a, b, c\}, \emptyset \longrightarrow_{(a,\{b,c\})} \emptyset, \{(a, \{b, c\})\}$ at the specification level. In general, the construction of a tree $t$ will decompose in $2 * n(t)$ steps. As expected, the obtained code is not correct yet, and may well deadlock, as in the following where $\delta(a) = \delta(b) = 1$, and $\delta(c) = 3$:

$$\begin{aligned}
\text{NODE}_a \mid \text{NODE}_b \mid \text{NODE}_c \longrightarrow\ & \text{BUILD}^1_a \mid \text{WAIT}^1_{a\star} \mid \text{NODE}_b \mid \text{NODE}_c \\
\longrightarrow\ & \text{WAIT}^1_{a\star} \mid \text{NODE}_b \mid \text{BUILD}^2_c \mid \text{WAIT}^2_{ca} \\
\longrightarrow\ & \text{WAIT}^1_{a\star} \mid \text{WAIT}^0_{bc} \mid \text{BUILD}^1_c \mid \text{WAIT}^2_{ca} \\
\equiv\ & \text{WAIT}^1_{a\star} \mid \overline{w_c}.\ \uparrow^b_c \mid \text{BUILD}^1_c \mid w_c.w_c.\overline{w_a}.\ \uparrow^c_a\ u \\
\longrightarrow\ & \text{WAIT}^1_{a\star} \mid \uparrow^b_c \mid \text{BUILD}^1_c \mid w_c.\overline{w_a}.\ \uparrow^c_a
\end{aligned}$$

At this stage, the incoherent tree $(a, \{(c, \{b\})\})$ is built, but there is no node left for $\text{BUILD}^1_c$ to recruit. Yet there is a successful trace, where $a$ recruits $b$ instead of $c$, corresponding at the specification level to the single transition $\{a, b, c\}, \emptyset \longrightarrow_{(a,\{b\})} \{c\}, \{(a, \{b\})\}$.

Therefore, it is clearly impossible to exhibit a bisimulation relation between the specification and the code induced LTS. However, the code is correct in

$$m ::= \langle\rangle \mid \langle i\rangle.m \mid \langle \theta, \alpha, p\rangle.m \mid \langle\!|\theta|\!\rangle.m$$
$$r, s ::= m \rhd p \mid (r \mid s) \mid (x)r$$
$$m \rhd (p \mid q) \equiv \langle 1\rangle.m \rhd p \mid \langle 2\rangle.m \rhd q$$
$$m \rhd (a)p \equiv (a)(m \rhd p) \text{ if } a \notin m$$

Fig. 3. RCCS memories, terms and additional congruence rules.

the weaker sense that its causal computations (defined below) indeed match the specification. As we will see in the next section this is enough to ensure correctness, provided the process is re-interpreted in RCCS. The idea is that, for instance, the deadlocked trace above may backtrack in RCCS up until the wrong decision of recruiting $c$ was made, and eventually recruit $b$. Note that this is not saying that the process will find a solution, it may well loop infinitely. There are known theoretical results showing that one cannot do better in a purely non-deterministic interpretation [14]. This is of little practical importance, since such backtracking schemes will be implemented with probabilities, and such futile infinite loops will have probability zero.

To prevent backtracking from a successful state, where a coherent tree has been constructed, the corresponding underlined final actions $\underline{ok_i}$ will be chosen irreversible.

## 4 Correctness

This section reviews the implementation of distributed backtracking in RCCS, and the reinterpretation theorem used to derive correctness of the previous section code.

### *4.1 RCCS*

RCCS is an extension of CCS where processes are equipped with memories used to undo computations. Memories and terms are given in Fig. 3 where: $i = 1, 2$; $\theta$ is an abstract name, drawn from a countable set $\mathcal{I}$, used to uniquely identify a communication; and $p$ is a CCS process (as in Sec. 3) with some distinguished underlined actions declared as irreversible.

In addition to the congruence rules (see Fig. 3) for distributing memories among forking processes, and commuting restrictions with memories (assuming $a$ was never used in the past –which is always possible using $\alpha$-conversion), product and sum are considered commutative and associative, and having 0 as neutral element, as in CCS.

Define $\mathcal{I}(m)$ (resp. $\mathcal{I}(r)$) to be the set of identifiers occurring in the memory $m$ (resp. memories of subprocesses of $r$). The RCCS labelled transition system

$$\text{act}\ \frac{\theta \notin \mathcal{I}(m)}{m \rhd \alpha.p + q \rightarrow_{\theta:\alpha} \langle \theta, \alpha, q \rangle.m \rhd p} \qquad \text{act-}\ \frac{\theta \notin \mathcal{I}(m)}{\langle \theta, \alpha, q \rangle.m \rhd p \rightarrow_{\theta:\alpha-} m \rhd \alpha.p + q}$$

$$\underline{\text{act}}\ \frac{\theta \notin \mathcal{I}(m)}{m \rhd \underline{\alpha}.p + q \rightarrow_{\theta:\underline{\alpha}} \langle \theta \rangle.m \rhd p}$$

$$\text{com}\ \frac{r \rightarrow_{\theta:\alpha} r' \quad s \rightarrow_{\theta:\alpha} s'}{r \mid s \rightarrow_{\theta:\tau} r' \mid s'} \qquad \text{com-}\ \frac{r \rightarrow_{\theta:\alpha-} r' \quad s \rightarrow_{\theta:\alpha-} s'}{r \mid s \rightarrow_{\theta:\tau-} r' \mid s'}$$

$$\underline{\text{com}}\ \frac{r \rightarrow_{\theta:\underline{\alpha}} r' \quad s \rightarrow_{\theta:\underline{\alpha}} s'}{r \mid s \rightarrow_{\theta:\underline{\tau}} r' \mid s'}$$

$$\text{par}\ \frac{r \rightarrow_{\theta:\zeta} r' \quad \theta \notin \mathcal{I}(s)}{r \mid s \rightarrow_{\theta:\zeta} r' \mid s} \qquad \text{res}\ \frac{r \rightarrow_{\theta:\zeta} r' \quad a \notin \zeta}{(a)r \rightarrow_{\theta:\zeta} (a)r'} \qquad \text{cgr}\ \frac{r_1 \equiv r \rightarrow_{\theta:\zeta} r' \equiv r_2}{r_1 \rightarrow_{\theta:\zeta} r_2}$$

Fig. 4. RCCS labelled transition system.

is given Fig. 4. Its labels are of the form $\theta : \zeta$, with $\zeta ::= \alpha \mid \alpha^- \mid \underline{\alpha}$, and $\theta$ an identifier. Side conditions of the form $\theta \notin \mathcal{I}(s)$ ensure $\theta$ is indeed unique (or a nounce in the cryptographic protocols terminology).

Forward action and communication rules each have their opposite, allowing to backtrack actions, unless the action is underlined, and thus explicitely made unbacktrackable.

Using abstract identifiers for uniquely tagging communication makes the presentation notably simpler, than in the original presentation [4], where a different scheme, more adapted to the theoretical study of RCCS was used. The abstract scheme is close the communication keys used in ref. [15]. Both schemes are shown equivalent in the appendix.

### *4.2 Reinterpretation theorem*

As said, the weaker notion of correction we need, uses the notion of causal trace. Intuitively, such traces do not involve contention among agents, since all actions therein contribute to the last one, and in addition represent atomic successful computations, since one asks the last action to be the trace only irreversible one.

More precisely, a trace $\sigma$ is said to be *causal* if it contains a single irreversible transition $\underline{t}$ and for all $\sigma' \sim \sigma$, $\sigma'$ ends by $\underline{t}$, where $\sim$ is the equivalence relation over CCS traces obtained by permuting concurrent transitions [1].

Here are some examples:

$$a.\underline{b}.0 \mid c.0 \;\rightarrow_a\; \underline{b}.0 \mid c.0 \;\rightarrow_c\; \underline{b}.0 \;\rightarrow_{\underline{b}}\; 0$$
$$a.\underline{b}.0 \mid c.0 \;\rightarrow_a\; \underline{b}.0 \mid c.0 \;\rightarrow_{\underline{b}}\; c.0$$
$$a.\underline{b}.0 \mid \bar{a}.0 \;\rightarrow_\tau\; \underline{b}.0 \;\rightarrow_{\underline{b}}\; 0$$

The first trace is not causal since its last action $\underline{b}$ commutes to the earlier action $c$, as in the second one which is causal; likewise, the last trace is causal, since the marked action $\underline{b}$ does not commute to $\tau$.

**Definition 4.1** Let $P$ be the set of CCS processes, $K$ be the set of underlined CCS actions, and define $p_1 \rightarrow^c_{\underline{k}} p_2$, if there is a causal trace from $p_1$ to $p_2$ ending with $\underline{k}$.

The *causal transition system* induced by $p$, written $CTS(p)$, is defined as $(P, p, K, \rightarrow^c_k)$.

In the examples above, one has $a.\underline{b}.0 \mid c.0 \rightarrow^c_{\underline{b}} c.0$, $a.\underline{b}.0 \mid \bar{a}.0 \rightarrow^c_{\underline{b}} 0$, and *not* $a.\underline{b}.0 \mid c.0 \rightarrow^c_{\underline{b}} 0$.

The theorem below asserts that the LTS induced by the interpretation of $p$ in RCCS is equivalent to $CTS(p)$, when observations are restricted to irreversible actions.

**Theorem 4.2 ([4])** *Let $p$ be a CCS process, and $\Phi$ be the relation $\{(\underline{k}, \theta : \underline{k}); \underline{k} \in K, \theta \in \mathcal{I}\}$, then $CTS(p) \approx_\Phi LTS(\langle\rangle \rhd p)$.*

### *4.3 Back to self assembling trees*

To apply this definition to the case of interest, we need to map macro-states (states of the specification) to micro-states (states of the corresponding process). Define first the family of maps $[\![\_]\!]_\alpha$, with $\alpha \in V + \{\star\}$:

$$[\![(a, \{t_1, \ldots, t_n\})]\!]_\alpha = \uparrow^a_\alpha \mid [\![t_1]\!]_a \mid \ldots \mid [\![t_n]\!]_a$$

This obtains a map from macro-states to what one might call their standard representation as micro-states (restrictions are not shown):

$$[\![N, \textstyle\sum_i t_i]\!] = \prod_{i \in N} \textsc{node}_i \mid \prod_j [\![t_j]\!]_\star$$

Defining $\Phi' = \{(t, \underline{ok_i}) \mid i \in V\}$, one has:

**Proposition 4.3** *The relation $\{(N, \sum_i t_i), [\![N, \sum_i t_i]\!]\}$ is a $\Phi'$-bisimulation between the specification LTS and $CTS([\![V]\!])$.*

The proof is routine. Concretely, this is saying two things. Firstly, whenever some tree may be constructed from the remaining free nodes of the specification, there exists a causal sequence of interactions among the agents that

implements it (see first example in Sec. 3). Secondly, whenever a tree is built after a successful series of agent interactions, this tree is indeed coherent, and therefore corresponds to a transition in the specification (this is even easier to prove, since the number of neighbours of any given process representing a node is always kept smaller or equal to its arity as specified by $\delta$).

Putting that proposition together with the theorem above one obtains:

**Corollary 4.4** *The specification LTS and* $LTS(\langle\rangle \rhd [\![V]\!])$*, are* $\Phi';\Phi$*-bisimilar.*

One may object that the visibility relation $\Phi';\Phi$ used here is highly non-injective, since it relates a tree $t$ to some $ok_i$, which contains no other information than the name of the process being the root of $t$. Using a value-passing version of CCS, one can decorate the implementation and construct during the assembly an expression describing the tree being constructed, which could then be used to encode injectively $t$ in the final irreversible action concluding the construction. However, the bisimulation relation we exhibit clearly contains all the needed information since the macro-to-micro map itself is injective.

## 5 Discussion

It remains to appreciate whether a direct solution in CCS could compare well with the indirect solution we have obtained. We base our discussion on a comparison with one particular reasonable direct implementation, given Fig. 5, and obtained by patching the indirect code to recover from deadlocks. The recruitment phase is quite similar to the one in the previous code, except BUILD and WAIT processes are now run in sequence. A more important difference is that the root may abort the construction by running at any time the process $\text{ABORT}_i^S$ which waits for the $\text{FREE}^S(end)$ process to free recruited agents, and then re-spawns the initial state. Any already recruited agent $i$ enters the abort state upon reception of a request by its parent using action $kill_i$. Accordingly, the final state $\uparrow_{i\alpha}^S$ indicating that the $i^{th}$ agent has finished its part of the recruitment, in the case $\alpha \neq \star$ still waits for a possible such abort request initiated by the root agent and forwarded by its parent.

Thus, the direct code may escape deadlocks. To keep things simple, we give up part of the distributed structure of the system: a node does not wait for the confirmations of its children until it has completed its recruiting task. This results in a better control of the construction process at the price of a loss of efficiency, since no agent can validate its recruitment until its parent is ready to receive the validation. Yet the main difference is in the backtracking mechanism: the RCCS code finds its way to a final shape by using partial backtracking, whereas the CCS one uses a top-down cancellation procedure to abort altogether the construction (as in ref. [9]).

One sees the RCCS code is more intuitive; this is because, in essence, it is easier to describe what has to be done, than what has to be undone.

$$
\begin{aligned}
\textsc{node}_i &=_{def} \tau.\textsc{build}_{i\star}^{\delta(i),\emptyset} + \sum_{j\in I} r_{ij}.\textsc{build}_{ij}^{\delta(i),\emptyset} \\
\textsc{build}_{ij}^{n+1,S} &=_{def} \sum_{k\in I} \bar{r}_{ik}.\textsc{build}_{ij}^{n,S\cup\{k\}} + kill_i.\textsc{abort}_i^S \\
\textsc{build}_{i\star}^{n+1,S} &=_{def} \sum_{k\in I} \bar{r}_{ik}.\textsc{build}_{ij}^{n,S\cup\{k\}} + \tau.\textsc{abort}_i^S \\
\textsc{build}_{i\alpha}^{0,S} &=_{def} \textsc{wait}_{i\alpha}^{|S|,S} \\
\textsc{wait}_{ij}^{n+1,S} &=_{def} w_i.\textsc{wait}_{ij}^{n,S} + \ kill_i.\textsc{abort}_i^S \\
\textsc{wait}_{i\star}^{n+1,S} &=_{def} w_i.\textsc{wait}_{i\star}^{n,S} + \tau.\textsc{abort}_i^S \\
\textsc{wait}_{ij}^{0,S} &=_{def} \overline{w}_j.\uparrow_{ij}^S + kill_i.\textsc{abort}_i^S \\
\textsc{wait}_{i\star}^{0,S} &=_{def} \underline{ok_i}.\uparrow_{i\star}^S \\
\textsc{free}^{S\cup\{i\}}(end) &=_{def} \overline{kill}_i.\textsc{free}^S(end) \\
\textsc{free}^{\emptyset}(end) &=_{def} \overline{end}.0 \\
\uparrow_{ij}^S &=_{def} \tau.\uparrow_{ij}^S + kill_i.\textsc{abort}_{ij}^S \\
\uparrow_{i\star}^S &=_{def} \tau.\uparrow_{i\star}^S \\
\textsc{abort}_i^S &=_{def} (end)(\textsc{free}^S(end) \mid end.\textsc{n}_i)
\end{aligned}
$$

Fig. 5. Self-assembly directly in CCS.

Furthermore, it is necessary to prove that the complete code conforms to its specification, and exhibit a bisimulation relation between the code and the specification (given Sec. 2). It is not clear at all how to do this by hand, and to get a sense of how difficult that may be, we have tested our code with the Mobility Workbench [16], a toolkit able to verify certain properties on $\pi$-calculus [13] processes. We succeeded in building the bisimulation relation for a system composed of 3 agents. For such a simple system, the Mobility Workbench already returns 600 states. Running the tool for 24 hours was not enough to obtain an answer in the case of a system of 4 agents.[2] The reason for this explosion in the size of the bisimulation is that the backtracking mechanism induces a lot of transitory states that try to undo their local constructions. More details about how the indirect method helps in automated verification can be found in ref. [11].

## 6 Conclusion and future work

We have presented a distributed algorithm for self assembling trees using CCS. Part of the appeal of the solution is that both the language used and the solu-

[2] Tests were made with a 1.4 GHz Pentium M with 256 MB of RAM.

tion itself stay simple. First one formulates a solution which is only required to be correct in weak sense. One then uses the reversible infrastructure provided by RCCS to obtain correctness. Not only the proof is greatly simplified in so doing, but the actual code obtained is also simpler in that backtracking stays implicit.

Our model leaves aside more subtle forms of self-assembly based on graph-rewriting. These would likely need a more powerful language [7,6], but there seems to be no reason why the decomposition of the self-assembly question advocated in this paper, would not extend to these richer languages. Our model also ignores the question of how one represents real space, in that connections are represented abstractly as synchronisations. Another important aspect of self-assembly which our model does not take into account is its quantitative nature, as our model only knows of non-deterministic evolutions, and doesn't assign to them any measure of their likelihood. More work is needed to understand how both spatial and probabilistic features could be added to the picture. One could think of a distributed language where agents would use timeouts to decide to backtrack. Substituting the RCCS operational semantics to the ordinary CCS one, or whichever richer language one is using, would obtain agents that would behave correctly with respect to the global specification. This requires first a thorough study of the impact of timeouts on the operational semantics of RCCS, a question which we plan to address in future work.

Decoupling in a given system the forward and backward components of its behaviour, is even more natural in the modelling and analysis of biomolecular interactions. Indeed, one may regard molecules as blind agents trying to bind haphazardly. Each time their spatial configurations match, proteins have a chance to bind, and these bounds are also frequently broken down. These exploration mechanisms have been argued to be of central importance in the evolvability of biological systems [8]. Here the implicit backtracking mechanism of RCCS comes in handy as a transparent way to model this instability [2], but, if anything, the addition of probabilities to backward moves, so as to generate a quantitative behaviour and be able to tune the backtracking mechanism, seems even more important in this specific context, and it remains to be seen how the method we have illustrated here can cope with these.

## References

[1] Boudol, G. and I. Castellani, *Permutation of transitions: An event structure semantics for CCS and SCCS*, in: *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS **354**, 1989, pp. 411–427.

[2] Danos, V. and J. Krivine, *Formal molecular biology done in CCS*, in: *Proceedings of BIO-CONCUR'03, Marseilles, France*, ENTCS (2003).

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

[4] Danos, V. and J. Krivine, *Transactions in RCCS*, in: *CONCUR'05*, LNCS **3653** (2005).

[5] Danos, V., J. Krivine and P. Sobocinski, *General reversibility*, in: *EXPRESS'06*, ENTCS (2006), to appear.

[6] Danos, V. and C. Laneve, *Formal molecular biology.*, TCS **325** (2004), pp. 69–110.

[7] Danos, V. and F. Tarissan, *Self-assembling graphs*, in: J. Mira and J. Alvarez, editors, *IWINAC'05*, LNCS **3561** (2005), pp. 501–510.

[8] Kirschner, M. and J. Gerhart, *Evolvability*, PNAS **95** (1998), pp. 8420–8427.

[9] Klavins, E., *Automatic synthesis of controllers for assembly and formation forming*, in: *ICRA'02*, 2002.

[10] Krivine, J., "Algèbres de Processus Réversibles," Ph.D. thesis, Université Paris 6 & INRIA-Rocquencourt (2006), moscova.inria.fr/~krivine.

[11] Krivine, J., *A verification algorithm for declarative concurrent programming*, Technical report, INRIA-Rocquencourt (2006), moscova.inria.fr/~krivine.

[12] Milner, R., "Communication and Concurrency," International Series on Computer Science, Prentice Hall, 1989.

[13] Milner, R., J. Parrow and D. Walker, *A calculus of mobile process (*i *and* ii*)*, Information and Computation **100** (1992), pp. 1–77.

[14] Palamidessi, C., *Comparing the expressive power of the synchronous and asynchronous pi-calculi*, MSCS **13** (2003), pp. 685–719.

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

[16] Victor, B. and F. Moller, *The Mobility Workbench — a tool for the π-calculus*, in: D. Dill, editor, *CAV'94*, LNCS **818** (1994), pp. 428–440.

## Appendix

Instead of abstract names, one can use memories as concrete identifiers [3]. We recall in this appendix how this is done, and argue that both the abstract and concrete identifying schemes are in fact intertranslatable. This is useful in so far as the reinterpretation theorem we used earlier was actually proven only for the concrete scheme. A complete proof is in ref. [10, Chap. 3].

Concrete memories are given as:

$$m ::= \langle\rangle \mid \langle i\rangle \cdot m \mid \langle \star, \alpha, p\rangle \cdot m \mid \langle m', \alpha, p\rangle \cdot m \mid \langle\!\langle \circ \rangle\!\rangle \cdot m$$

where $\star$ stands for an unknown communication partner, the equivalent of which, in the semantics above, is a $\theta$ that is unique to the whole process. The corresponding transition system, shown below, has now labels of the form $\mu : \zeta$ where $\mu$ is a set of one or two memories; $r_{m'@m}$ denotes the substitution of $\star$ with the concrete identifier $m'$ in $\langle \star, \alpha, p\rangle \cdot m$; irreversible rules are not shown.

$$\frac{}{m \rhd \alpha.p + q \to_{m:\alpha} \langle \star, \alpha, q\rangle \cdot m \rhd p} \qquad \frac{}{\langle \star, \alpha, q\rangle \cdot m \rhd p \to_{m:\alpha^-} m \rhd \alpha.p + q}$$

$$\frac{r \to_{m:\bar{a}} r' \quad s \to_{m':a} s'}{r \mid s \to_{m,m':\tau} r'_{m'@m} \mid s'_{m@m'}} \qquad \frac{r \to_{m:\bar{a}^-} r' \quad s \to_{m':a^-} s'}{r_{m'@m} \mid s_{m@m'} \to_{m,m':\tau^-} r' \mid s'}$$

$$\frac{r \to_{\mu:\zeta} r'}{r \mid s \to_{\mu:\zeta} r' \mid s} \qquad \frac{r \to_{\mu:\zeta} r' \quad \zeta \neq a, \bar{a}, a^-, \bar{a}^-}{(a)r \to_{\mu:\zeta} (a)r'} \qquad \frac{r \equiv r_1 \to_{\mu:\zeta} r_2 \equiv r'}{r \to_{\mu:\zeta} r'}$$

Given an abstract process $r$, and assuming any identifier occurs at most twice in $r$, the following defines inductively a map $M_r$ from an abstract process to a concrete one (all other clauses being trivial):

$$M_r(\langle\!|\theta|\!\rangle \cdot m) = \langle\!|\circ|\!\rangle \cdot M_r(m)$$

$$M_r(\langle \theta, \alpha, p\rangle \cdot m) = \begin{cases} \langle M_r(m'), \alpha, p\rangle \cdot M_r(m) & \text{if } \langle \theta, \bar{\alpha}, q\rangle \cdot m' \in r \\ \langle \star, \alpha, p\rangle \cdot M_r(m) & \text{else} \end{cases}$$

Conversely, given a $\mu$ indexed family of identifiers $\theta_\mu$ such that $\theta_\mu \neq \theta_{\mu'}$ if $\mu \cap \mu' \neq \mu$, one can map concrete processes to abstract ones (again all other clauses are trivial):

$$\Theta(\langle\!|\circ|\!\rangle \cdot m) = \langle\!|\theta_{\{m\}}|\!\rangle \cdot \Theta(m)$$

$$\Theta(\langle m, \alpha, p\rangle \cdot m') = \langle \theta_{\{m,m'\}}, \alpha, p\rangle \cdot \Theta(m')$$

$$\Theta(\langle \star, \alpha, p\rangle \cdot m) = \langle \theta_{\{m\}}, \alpha, p\rangle \cdot \Theta(m)$$

We suppose now all concrete processes have unique memories, and all abstract processes have identifiers occurring at most twice. This is easily shown to be preserved under computations.

**Proposition 6.1** *If $r \to_{\theta:\zeta} s$ then $\exists \mu : M_r(r) \to_{\mu:\zeta} M_s(s)$ and if $r \to_{\mu:\zeta} s$ then $\exists \theta : \Theta(r) \to_{\theta:\zeta} \Theta(s)$.*

For the first implication: if $r \to_{\theta:\tau} s$, take $\mu = \{M_s(m_1), M_s(m_2)\}$ where $\langle \theta, \alpha, p\rangle \cdot m_1$, $\langle \theta, \bar{\alpha}, q\rangle \cdot m_2 \in s$; if $r \to_{\theta:\tau^-} s$, take $\mu = \{M_r(m_1), M_r(m_2)\}$ where $\langle \theta, \alpha, p\rangle \cdot m_1$, $\langle \theta, \bar{\alpha}, q\rangle \cdot m_2 \in r$. For the second implication, it suffices to take $\theta = \theta_\mu$. The side condition in the par rule (see Fig. 4) holds thanks to the unicity of memories and the assumption that $\theta_\mu \neq \theta_{\mu'}$ whenever $\mu \cap \mu' \neq \mu$.