Bisimulation by Unification*

Paolo Baldan[1], Andrea Bracciali[2], and Roberto Bruni[2]

[1] Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italia.
[2] Dipartimento di Informatica, Università di Pisa, Italia.
baldan@dsi.unive.it, braccia@di.unipi.it, bruni@di.unipi.it

Abstract We propose a methodology for the analysis of open systems based on process calculi and bisimilarity. Open systems are seen as coordinators (i.e. terms with place-holders), that evolve when suitable components (i.e. closed terms) fill in their place-holders. The distinguishing feature of our approach is the definition of a symbolic operational semantics for coordinators that exploits spatial/modal formulae as labels of transitions and avoids the universal closure of coordinators w.r.t. all components. Two kinds of bisimilarities are then defined, called *strict* and *large*, which differ in the way formulae are compared. Strict bisimilarity implies large bisimilarity which, in turn, implies the one based on universal closure. Moreover, for process calculi in suitable formats, we show how the symbolic semantics can be defined constructively, using unification. Our approach is illustrated on a toy process calculus with CCS-like communication within ambients.

Introduction

The ever increasing usage and development of mobile devices raise the need of formal models for open systems, where components can be dynamically connected to interact with network services. Process calculi (PC) are often instrumental in focusing on certain aspects like communications, distribution and causal dependencies. However, PC techniques are mostly devised for the study of *components* (i.e. closed terms of the calculus) rather than *coordinators* (i.e. contexts with holes marked by process variables).

In particular, while the operational semantics and several equivalences have been often defined for components (e.g., based on either bisimulation or traces or testing), their extensions to coordinators usually require additional efforts. Roughly, an equivalence $\approx$ defined on components can be lifted to coordinators by letting $C[X_1,\ldots,X_n] \approx D[X_1,\ldots,X_n]$ when $C[p_1,\ldots,p_n] \approx D[p_1,\ldots,p_n]$ for all components $p_1,\ldots,p_n$. In the case of bisimulation, this means that the coalgebraic techniques applicable to components fall short for coordinators, since the definition involves universal quantification on components. Instead, a symbolic technique for allowing contexts to "bisimulate without instantiation" would ease the analysis and verification of coordinators' properties.

This issue finds its dual formulation in the contextual closure needed when the bisimilarity on components $\sim$ is not a congruence and one defines the largest congruence $\simeq$ contained in $\sim$ (by letting $p \simeq q$ if for all contexts $C[.]$, identity included, $C[p] \sim C[q]$ holds). Note that in general $\simeq$ is not a bisimulation. The largest congruence

* Research supported by the IST programme on FET-GC Projects AGILE, MYTHS and SOCS.

which is also a bisimulation is called *dynamic bisimilarity* and it is defined by allowing context closure at each bisimulation step [22].

To avoid universal quantification on contexts, several authors—e.g., Sewell in [25], Leifer and Milner in [20]—propose a symbolic transition system for components whose labels are the "minimal" contexts needed to the component for evolving. A transition

$$p \xrightarrow{C[\cdot,X_1,\cdots,X_n]} D[X_1,\ldots,X_n]$$

means that $C[p,p_1,\ldots,p_n]$ can reduce in one step to $D[p_1,\ldots,p_n]$, and that C is strictly necessary to perform the step. However, in their symbolic systems, though transitions always depart from components, they may lead also to contexts (like D above) and therefore bisimulation must be defined on contexts via universal quantification over all possible closed instantiations. Thus, the problem of universal quantification is shifted from contexts to components. Finding a sound and efficient way to face this problem is the goal of our contribution.

Symbolic Bisimulation. It is nowadays commonly accepted that the operational semantics of most process calculi can be conveniently expressed by exploiting two basic ingredients, according to Plotkin's SOS recipe [23]: the structure of the components and the behaviour of their subcomponents. Thus, a PC definition usually involves a process signature Σ, a structural equivalence $\equiv$ on process terms, and a labelled transition system (LTS) specified through a set of inductive (structural) proof rules. [1]

It follows that the behaviour of a coordinator can depend: (1) on the spatial structure of the components that are inserted/substituted/connected in/with it; (2) on their behaviour, i.e. on the actions that can be observed.

The first attempt could be to define a transition system whose states are coordinators and whose arcs are labelled with the components that allow coordinators to evolve. But this would result in a too large transition system, making verification difficult.

To attack this problem, reducing the size of the transition system, we propose to borrow formulae from a suitable logic for expressing the most general class of processes with whom each coordinator can react. This leads us to the notion of *symbolic transition system* (STS), whose states are coordinators and whose transitions have the shape

$$C[X_1,\ldots,X_n] \xrightarrow{\varphi_1,\ldots,\varphi_n} D[Y_1,\ldots,Y_m]$$

meaning that the step can be performed by $C[p_1,\ldots,p_n]$ whenever $p_i \models \varphi_i$, for $i \in [1,n]$.

The logic where the formulae φ_i's live and the notion of satisfaction $\models$ must be of course targeted to the PC under study. In general, the logic may involve both spatial and temporal aspects of components, i.e. it can be a *spatial logic* [8,11].

Fixed an STS, two kinds of bisimilarities $\sim_{\text{strict}}$ and $\sim_{\text{large}}$, referred to, respectively, as *strict* and *large*, can be defined on coordinators, differing for the way labels (i.e. formulae) are compared, with $\sim_{\text{strict}} \Rightarrow \sim_{\text{large}}$, and $\sim_{\text{strict}}$ being an equivalence relation. We show that, whenever the STS satisfies suitable properties of correspondence w.r.t. the operational semantics of components, called *correctness* and *completeness*, $\sim_{\text{large}}$ (and thus $\sim_{\text{strict}}$) implies the equivalence induced by the universal closure.

[1] Reduction semantics can be obviously recasted in LTS's as the special case with a unique label.

For PC whose rules are in a quite general format, called *algebraic format* [16], we provide a constructive way of defining a spatio-temporal logic and we give an algorithm for building a correct and complete STS over such a logic. The algorithm, expressed as a Prolog program, builds labels by computing recursively the most general unifiers between coordinators and left-hand sides of the operational rules.

Synopsis. In § 1 we fix the notation and we recall some basic definitions. In § 2, we overview the general ideas on which our approach relies, introducing the notion of (correct and complete) STS, defining large and strict symbolic bisimilarities and showing that both relations imply bisimilarity via universal closure. In § 3, first we illustrate the algorithmic construction of correct and complete STS's for process calculi with no structural axioms and operational rules in the *algebraic format* of [16], and then we show how to deal with the common AC1 axioms for the parallel composition operator. In § 4, we test our approach against a simple case study consisting of a fragment of the ambient calculus with CCS-like communication within ambients.

Related work. The aforementioned papers by Sewell, Leifer and Milner have motivated and inspired our quest for a set of labels powerful enough to model the maximal classes of components with whom coordinators can react. The papers by Caires, Cardelli and Gordon on spatial logics have suggested us an elegant mathematical tool for expressing both structural and temporal constraints in the labels. It is worth mentioning that spatial formulae have many analogies with the topological modalities introduced separately by Fiadeiro *et al.* in [15] when proposing a verification logic for rewriting logic.

A symbolic approach to bisimulation in the case of value passing calculi, where actions are parametrised over a possibly infinite set, has been explored in [17].

Among other frameworks where the semantics of components and coordinators is defined uniformly, let us mention *tile logic* (TL) [16,6], *conditional transition systems* (CTS) [24] and *context systems* (CS) [19], which come also equipped with different formats for guaranteeing that bisimilarity is a congruence. While models based on TL, CTS and CS can be easily translated in our framework, the use of spatial formulae makes our approach applicable to a wider class of calculi.

The idea of using unification for building formulae comes from Logic Programming and more precisely by its view as an interactive system presented in [7].

1 Notation

To ease the presentation we consider one-sorted signatures, but our results easily extend to the many-sorted case. A *signature* is a set of *operators* Σ together with an arity function $ar : \Sigma \to \mathbb{N}$. For $n \in \mathbb{N}$, we let $\Sigma_n = \{f \in \Sigma \mid ar(f) = n\}$. We denote by $\mathbb{T}_\Sigma(\mathcal{X})$ the term algebra over Σ and variables in the set $\mathcal{X}$ (disjoint from Σ), with $\mathbb{T}_\Sigma = \mathbb{T}_\Sigma(\emptyset)$. For $P \in \mathbb{T}_\Sigma(\mathcal{X})$ we denote by $var(P)$ the set of variables $X \in \mathcal{X}$ that appear in P. If $var(P) = \emptyset$ then P is called *closed*, otherwise *open*. When signatures are used for presenting the syntax of PC, closed terms define the set $\mathcal{P}$ of *components* p of the calculus, while the general, possibly open, terms form the set $\mathcal{C}$ of *coordinators* C. Often we shall write $C[X_1, \ldots, X_n]$ to mean that C is a coordinator such that $var(C) \subseteq \{X_1, \ldots, X_n\}$.

$$\frac{P \equiv Q \in E,\ \sigma(P), \sigma(Q) \in \mathbb{T}_\Sigma}{\sigma(P) \equiv \sigma(Q)}\ (subs) \qquad \frac{p_1 \equiv q_1,\ \cdots,\ p_n \equiv q_n,\ f \in \Sigma_n}{f(p_1, \ldots, p_n) \equiv f(q_1, \ldots, q_n)}\ (context)$$

$$\frac{p \in \mathbb{T}_\Sigma}{p \equiv p}\ (refl) \qquad \frac{p \equiv q}{q \equiv p}\ (symm) \qquad \frac{p_1 \equiv p_2,\ p_2 \equiv p_3}{p_1 \equiv p_3}\ (trans)$$

Figure 1. Closure of structural axioms.

A *structural axiom* is a sentence $P \equiv Q$ for $P, Q \in \mathbb{T}_\Sigma(\mathcal{X})$. Given a set $E = \{P_i \equiv Q_i \mid i \in I\}$ of structural axioms, we say that a Σ-algebra $\mathbf{A}$ *satisfies* E if for any assignment $\sigma : \mathcal{X} \to \mathbf{A}$ of values to the variables in $\mathcal{X}$, we have that $\sigma(P_i) =_{\mathbf{A}} \sigma(Q_i)$, for all $i \in I$. The initial algebra $\mathbb{T}_{\Sigma,E}$ is the quotient of $\mathbb{T}_\Sigma$ modulo the equivalence $\equiv$ defined in Fig. 1, where axioms in E are closed w.r.t. substitution, contextualization, reflexivity, symmetry, and transitivity.

Process calculi come often equipped with LTS operational semantics, where states are components over the process signature Σ, labels range over a suitable alphabet Λ, and transitions model the activities of components. Commonly such LTS is specified by a collection of inductive (transition) proof rules. In the presence of structural axioms, states are equivalence classes of components (modulo $\equiv$), as if proof rules included:

$$\frac{p' \equiv p \qquad p \xrightarrow{a} q \qquad q' \equiv q}{p' \xrightarrow{a} q'}\ (equiv)$$

A *bisimulation* is a symmetric, reflexive relation $\approx$ over components such that if $p \approx q$, then for any transition $p \xrightarrow{a} p'$ there exists a component q' and a transition $q \xrightarrow{a} q'$ with $p' \approx q'$. We denote by $\sim$ the largest bisimulation and call it *bisimilarity*. Note that the rule $(equiv)$ makes $p \sim q$ hold whenever $p \equiv q$. We call *universal bisimilarity* the usual lifting of $\sim$ to coordinators obtained by closing for all possible substitutions:

$$C[X_1, \ldots, X_n] \sim D[X_1, \ldots, X_n] \overset{\text{def}}{\iff} \forall p_1, \ldots, p_n \in \mathcal{P}, \quad C[p_1, \ldots, p_n] \sim D[p_1, \ldots, p_n]$$

2 Formulae as Labels

The definition of bisimilarity on coordinators based on the closure with respect to any possible substitution presents obvious drawbacks. In fact, to verify the bisimilarity of two coordinators, one is typically led to check the bisimilarity of infinitely many processes (all the possible closed instances of the coordinators). Furthermore bisimilarity of coordinators is not defined in a coinductive way and thus the coalgebraic techniques applicable to components fall short for coordinators. In trying to prove the equivalence of two coordinators it is thus convenient to perform a kind of symbolic calculation:

1. without instantiating components which do not play an active role in a step and instantiating the active components as little as possible;
2. making assumptions not only on the structure, but (as in TL, CTS, CS) also on the behaviour of the active components.

The above strategy is formalised by introducing a symbolic transition system whose states are coordinators and whose labels encode the structural and/or behavioural conditions (see points 1–2 above) that components should fulfill for enabling the move.

In the following, we assume that a process calculus PC is fixed with signature Σ and structural axioms E, whose semantics is given by the LTS $\mathcal{L}$ over $\mathbb{T}_{\Sigma,E}$ and label alphabet Λ. We also assume that a logic L over components is given, which may have modal operators and whose atomic formulae include the process variables in $\mathcal{X}$ and the components in $\mathcal{P}$ (with $p \models X$ for any $p \in \mathcal{P}$ and $X \in \mathcal{X}$, while $p \models q$ iff $p \equiv q$ for any $p, q \in \mathcal{P}$, where $\models$ is satisfaction).

Definition 1 (Symbolic Transition System). *A* symbolic transition system *(*STS*)* $\mathcal{S}$ *over* L *for the process calculus PC is a set of transitions*

$$C[X_1,\ldots,X_n] \stackrel{(\varphi_1,\ldots,\varphi_n)}{\longrightarrow}_a D[Y_1,\ldots,Y_m]$$

where $C[X_1,\ldots,X_n]$ *and* $D[Y_1,\ldots,Y_m]$ *are coordinators,* $a \in \Lambda$ *and* φ_i *are formulae in* L *containing only variables from* $\{Y_1,\ldots,Y_m\}$.

The variable names in the states of $\mathcal{S}$ are not relevant: they are just indexed placeholders, whose number can vary along the computation. The correspondence between variables in the source (e.g. X_i) and their residuals (e.g. Y_j) in the target is expressed by the formulae (e.g. φ_i), in which the residuals may occur. For example, the modal formula $\varphi_i = \diamond a.Y_j$ is satisfied by any process performing a (and its residual replaces Y_j in D).

For $\mathcal{S}$ to be an abstract view of PC we must of course require some additional properties enforcing the correspondence with the concrete LTS $\mathcal{L}$. Consider a transition system where coordinators have just one hole. Intuitively, whenever $C[X] \stackrel{\varphi}{\longrightarrow}_a D[Y]$ the idea is that the coordinator C, when instantiated with any component satisfying φ, can perform action a becoming an instance of D. The process variable Y, which typically occur in φ, is intended to represent the residual of what substituted for X, after it has exhibited the capabilities required by φ. More precisely, for any component q such that $p \models \varphi[q/Y]$ (where $\varphi[q/Y]$ denotes the formula obtained from φ by replacing all the occurrences of Y by q) the component $C[p]$ can perform an action a becoming $D[q]$. On the other hand, any concrete transition on components should have symbolic counterparts. These two properties are formalised as *correctness* and *completeness*, respectively.

Definition 2 (Correctness). *An* STS $\mathcal{S}$ *for the process calculus PC is* correct *if for any symbolic transition*

$$C[X_1,\ldots,X_n] \stackrel{(\varphi_1,\ldots,\varphi_n)}{\longrightarrow}_a D[Y_1,\ldots,Y_m]$$

in $\mathcal{S}$*, for any* $q_1,\ldots,q_m$ *and for any* $p_i \models \varphi_i[q_1/Y_1,\ldots,q_m/Y_m]$ *for* $i \in [1,n]$*, there exists a transition* $C[p_1,\ldots,p_n] \stackrel{a}{\longrightarrow} D[q_1,\ldots,q_m]$ *in* $\mathcal{L}$*.*

Definition 3 (Completeness). *An* STS $\mathcal{S}$ *for the process calculus PC is* complete *if for any coordinator* $C[X_1,\ldots,X_n]$*, for all components* $p_1,\ldots,p_n$ *and for any transition*

$$C[p_1,\ldots,p_n] \stackrel{a}{\longrightarrow} q$$

in $\mathcal{L}$ *there exists a transition* $C[X_1,\ldots,X_n] \stackrel{(\varphi_1,\ldots,\varphi_n)}{\longrightarrow}_a D[Y_1,\ldots,Y_m]$ *in* $\mathcal{S}$ *and* $q_1,\ldots,q_m$ *such that* $p_i \models \varphi_i[q_1/Y_1,\ldots,q_m/Y_m]$ *for* $i \in [1,n]$*, and* $q \equiv D[q_1,\ldots,q_m]$*.*

Over any STS we can straightforwardly define a bisimulation-like equivalence.

Definition 4 (Strict Symbolic Bisimulation). *A symmetric relation $\approx$ over the set of coordinators $\mathcal{C}$ is a* strict symbolic bisimulation *if for any two coordinators $C[X_1,\ldots,X_n]$ and $D[X_1,\ldots,X_n]$ such that $C[X_1,\ldots,X_n] \approx D[X_1,\ldots,X_n]$, for any transition*

$$C[X_1,\ldots,X_n] \xrightarrow{(\varphi_1,\ldots,\varphi_n)}_a C'[Y_1,\ldots,Y_m]$$

there exists a transition $D[X_1,\ldots,X_n] \xrightarrow{(\varphi_1,\ldots,\varphi_n)}_a D'[Y_1,\ldots,Y_m]$ such that $C'[Y_1,\ldots,Y_m] \approx D'[Y_1,\ldots,Y_m]$. The largest strict symbolic bisimulation is an equivalence relation called strict symbolic bisimilarity *and denoted by $\sim_{\text{strict}}$.*

Our first result states that the strict symbolic bisimilarity distinguishes as much as universal (closure) bisimilarity $\sim$, as defined by the end of Section 1.

Theorem 1 ($\sim_{\text{strict}} \Rightarrow \sim$). *If $\mathcal{S}$ is a correct and complete* STS*, then*

$$C[X_1,\ldots,X_n] \sim_{\text{strict}} D[X_1,\ldots,X_n] \;\Rightarrow\; C[X_1,\ldots,X_n] \sim D[X_1,\ldots,X_n]$$

Proof. Suppose $C[X_1,\ldots,X_n] \sim_{\text{strict}} D[X_1,\ldots,X_n]$. We want to show that for any $p_1,\ldots,p_n$, we have $C[p_1,\ldots,p_n] \sim D[p_1,\ldots,p_n]$. Let $\mathcal{R}_{\text{strict}}$ be the relation defined by

$$C[p_1,\ldots,p_n]\; \mathcal{R}_{\text{strict}}\; D[p_1,\ldots,p_n] \stackrel{\text{def}}{\Longleftrightarrow} C[X_1,\ldots,X_n] \sim_{\text{strict}} D[X_1,\ldots,X_n].$$

We first show that $\mathcal{R}_{\text{strict}}$ is a bisimulation for $\mathcal{L}$.

For any transition $C[p_1,\ldots,p_n] \xrightarrow{a} q$ in $\mathcal{L}$, by completeness of $\mathcal{S}$, a symbolic transition $C[X_1,\ldots,X_n] \xrightarrow{(\varphi_1,\ldots,\varphi_n)}_a C'[Y_1,\ldots,Y_m]$ and m components $q_1,\ldots,q_m$ exist such that $p_i \models \varphi_i[q_1/Y_1,\ldots,q_m/Y_m]$ and $q \equiv C'[q_1,\ldots,q_m]$. Since $C[X_1,\ldots,X_n] \sim_{\text{strict}} D[X_1,\ldots,X_n]$ by hypothesis, we have that $D[X_1,\ldots,X_n] \xrightarrow{(\varphi_1,\ldots,\varphi_n)}_a D'[Y_1,\ldots,Y_m]$ with $C'[Y_1,\ldots,Y_m] \sim_{\text{strict}} D'[Y_1,\ldots,Y_m]$. By correctness of $\mathcal{S}$, and by $p_i \models \varphi_i[q_1/Y_1,\ldots,q_m/Y_m]$ for all $i \in [1,n]$, it holds that $D[p_1,\ldots,p_n] \xrightarrow{a} D'[q_1,\ldots,q_m]$. Since $C'[Y_1,\ldots,Y_m] \sim_{\text{strict}} D'[Y_1,\ldots,Y_m]$, we have that $C'[q_1,\ldots,q_m]\; \mathcal{R}_{\text{strict}}\; D'[q_1,\ldots,q_m]$. The relation $\mathcal{R}_{\text{strict}}$ is obviously symmetric and hence it is a bisimulation. Since bisimilarity $\sim$ is the largest bisimulation, it contains $\mathcal{R}_{\text{strict}}$ and therefore $C[p_1,\ldots,p_m] \sim D[q_1,\ldots,q_m]$, concluding the proof. $\square$

2.1 Large Symbolic Bisimulation

The requirement of exact matching between formulae in the definition of strict symbolic bisimulation can be too strong, especially in the presence of spatial formulae and structural congruences. Hence, we propose a way to relax this condition.

To this aim, we assume that the logic L is a spatial logic whose operators include a subset Σ_{L} of Σ, with satisfaction defined by (for any $f \in \Sigma_{\mathsf{L}}$ with arity n):

$$p \models f(\varphi_1,\ldots,\varphi_n) \quad \text{iff } \exists p_1,\ldots,p_n.\ p \equiv f(p_1,\ldots,p_n) \wedge \forall i.\ p_i \models \varphi_i.$$

We call φ a *spatial formula* if it is built by using just variables $X \in \mathcal{X}$ and spatial operators $f \in \Sigma_{\mathsf{L}}$. Abusing the notation, a spatial formula can be either seen as a component/coordinator or as a logic formula, depending on the setting where it is used.

Definition 5 (Large Symbolic Bisimulation). *A symmetric relation $\approx$ over the set of coordinators $\mathcal{C}$ is a* large symbolic bisimulation *if for any pair of coordinators $C[X_1,\dots,X_n]$ and $D[X_1,\dots,X_n]$ such that $C[X_1,\dots,X_n] \approx D[X_1,\dots,X_n]$, for any transition*

$$C[X_1,\dots,X_n] \xrightarrow{(\varphi_1,\dots,\varphi_n)}_a C'[Y_1,\dots,Y_m]$$

a transition $D[X_1,\dots,X_n] \xrightarrow{(\psi_1,\dots,\psi_n)}_a D'[Z_1,\dots,Z_k]$ and k spatial formulae $\psi'_1,\dots,\psi'_k$ exist such that $\varphi_i = \psi_i[\psi'_1/Z_1,\dots,\psi'_k/Z_k]$ and $C'[Y_1,\dots,Y_m] \approx D'[\psi'_1,\dots,\psi'_k]$. The greatest large bisimulation is called large symbolic bisimilarity *and denoted $\sim_{\text{large}}$.*

Large symbolic bisimulation allows a transition to be simulated by another transition where the spatial constraints on the Y's are relaxed, so that "more general" components can be used for the X's. It follows that transitions in $\mathcal{S}$ that are dominated by transitions with a less (spatially) specified label can be abstracted away from the system.

Example 1. Let $\Sigma = \{a, f(.), g(.)\}$ and let the logic L include all the three corresponding spatial operators. Let $\mathcal{S}$ be the STS with transitions $f(X) \xrightarrow{X}_\tau X$, $g(X) \xrightarrow{X}_\tau X$, and $g(X) \xrightarrow{a}_\tau a$. Then it is obvious that $f(X) \not\sim_{\text{strict}} g(X)$, because the last transition of $g(X)$ cannot be matched by $f(X)$. However, the formula X is "more general" than the formula a, and therefore $f(X) \sim_{\text{large}} g(X)$. □

Remark 1. While $\sim_{\text{strict}}$ is an equivalence relation, we only proved that, if necessary, $\sim_{\text{large}}$ can be guaranteed to be an equivalence by suitably saturating the STS with redundant transitions. We also point out that the obvious way of relaxing the requirements of $\sim_{\text{strict}}$ by allowing a step $C[X] \xrightarrow{\varphi}_a C'[Y]$ to be simulated by $D[X] \xrightarrow{\psi}_a D'[Y]$ with $\varphi \Rightarrow \psi$, would not yield a consistent formulation, as it can be seen that, contrary to spatial operators, modal operators in φ cannot be safely abstracted away in ψ.

Proposition 1 **($\sim_{\text{strict}} \Rightarrow \sim_{\text{large}}$).** *For any symbolic transition system $\mathcal{S}$*

$$C[X_1,\dots,X_n] \sim_{\text{strict}} D[X_1,\dots,X_n] \;\Rightarrow\; C[X_1,\dots,X_n] \sim_{\text{large}} D[X_1,\dots,X_n].$$

Proof. It follows directly from the definition of the two bisimulations, since the spatial formulae ψ'_i's used in $\sim_{\text{large}}$ when simulating the step can of course be identities. □

Theorem 2 **($\sim_{\text{large}} \Rightarrow \sim$).** *If $\mathcal{S}$ is correct and complete w.r.t. $\mathcal{L}$, then*

$$C[X_1,\dots,X_n] \sim_{\text{large}} D[X_1,\dots,X_n] \;\Rightarrow\; C[X_1,\dots,X_n] \sim D[X_1,\dots,X_n].$$

Proof. The proof is similar to, but slightly more involved than, that of Theorem 1. Suppose $C[X_1,\dots,X_n] \sim_{\text{large}} D[X_1,\dots,X_n]$. We want to show that for any $p_1,\dots,p_n$, we have $C[p_1,\dots,p_n] \sim D[p_1,\dots,p_n]$. Let $\mathcal{R}_{\text{large}}$ be the relation defined by

$$C[p_1,\dots,p_n] \,\mathcal{R}_{\text{large}}\, D[p_1,\dots,p_n] \overset{\text{def}}{\iff} C[X_1,\dots,X_n] \sim_{\text{large}} D[X_1,\dots,X_n].$$

We first show that $\mathcal{R}_{\text{large}}$ is a bisimulation for $\mathcal{L}$. For any transition $C[p_1,\dots,p_n] \xrightarrow{a} q$ in $\mathcal{L}$, by completeness of $\mathcal{S}$, a symbolic transition $C[X_1,\dots,X_n] \xrightarrow{(\varphi_1,\dots,\varphi_n)}_a C'[Y_1,\dots,Y_m]$

and m components $q_1,\dots,q_m$ exist with $p_i \models \varphi_i[q_1/Y_1,\dots,q_m/Y_m]$ and $q \equiv C'[q_1,\dots,q_m]$. Since $C[X_1,\dots,X_n] \sim_{\text{large}} D[X_1,\dots,X_n]$ by hypothesis, we have

$$D[X_1,\dots,X_n] \stackrel{(\psi_1,\dots,\psi_n)}{\longrightarrow}_a D'[Z_1,\dots,Z_k]$$

and k spatial formulae $\psi'_1,\dots,\psi'_k$ exist such that $C'[Y_1,\dots,Y_m] \sim_{\text{large}} D'[\psi'_1,\dots,\psi'_k]$ and $\varphi_i = \psi_i[\psi'_1/Z_1,\dots,\psi'_k/Z_k]$ for all $i \in [1,n]$. Since $p_i \models \varphi_i[q_1/Y_1,\dots,q_m/Y_m]$ (for all $i \in [1,n]$), letting $q'_i \equiv \psi'_i[q_1/Y_1,\dots,q_m/Y_m]$, it follows that $p_i \models \psi_i[q'_1/Z_1,\dots,q'_k/Z_k]$. Therefore, by correctness of $\mathcal{S}$, it follows that $D[p_1,\dots,p_n] \xrightarrow{a} D'[q'_1,\dots,q'_k]$. Moreover, since $C'[Y_1,\dots,Y_m] \sim_{\text{large}} D'[\psi'_1,\dots,\psi'_k]$, we have that $C'[q_1,\dots,q_m]\ \mathcal{R}_{\text{large}}\ D'[q'_1,\dots,q'_k]$. The relation $\mathcal{R}_{\text{large}}$ is clearly symmetric and hence it is a bisimulation for $\mathcal{L}$. Since bisimilarity $\sim$ is the largest bisimulation, it contains $\mathcal{R}_{\text{large}}$ and thus $C[p_1,\dots,p_m] \sim D[q_1,\dots,q_m]$. $\square$

Note that Theorem 1 now follows as a corollary of Proposition 1 and Theorem 2.

3 Bisimulation by Unification

In this section we outline a methodology for deriving a correct and complete STS for algebraic PC, i.e. PC whose operational proof rules are in a quite general format, called *algebraic format* [16], recalled below. More specifically, given a PC, a logic L_{PC} with spatial and modal operators in the style of [8,11] can be systematically derived. Then the proof rules of the calculus are used to construct a Prolog program (finite if the set of proof rules of the PC is finite) which represents an STS over L_{PC} for the PC, in the sense that given any coordinator, the program allows to compute the set of its symbolic transitions. Such STS can be proved to be correct and complete for the given PC.

Definition 6 (Algebraic Format). *A proof rule is in* algebraic format *if it has the form*

$$\frac{\{X_i \xrightarrow{a_i} Y_i\}_{i\in I}}{C[X_1,\dots,X_n] \xrightarrow{a} D[Z_1,\dots,Z_n]}$$

with $I \subseteq [1,n]$, and where $Z_i = Y_i$ if $i \in I$ and $Z_i = X_i$ otherwise. An algebraic process calculus *is a* PC *whose proof rules are in algebraic format.*

The algebraic format generalises De Simone format [14] by allowing a generic context C, possibly involving more than one operator, (to appear) as left-hand side of the conclusion of the rule. However, it is worth recalling that while De Simone format guarantees that bisimilarity is a congruence, for algebraic PC's this is not necessarily the case.

3.1 A Spatio-Temporal Logic for Symbolic Transition Systems

Given a process calculus PC over a signature Σ we define the logic whose formulae will be used as labels in the STS. The logic must be powerful enough to be able to express, for any coordinator, the (more general) structural and behavioural properties which should be fulfilled by unspecified components to allow transitions to happen.

$$
\begin{array}{lll}
p \models X & & \\
p \models q & \text{iff} & p \equiv q \\
p \models \diamond a.\varphi & \text{iff} & \exists p'.\ p \xrightarrow{a} p' \ \wedge\ p' \models \varphi \\
p \models f(\varphi_1,\ldots,\varphi_n) & \text{iff} & \exists p_1,\ldots,p_n.\ p \equiv f(p_1,\ldots,p_n) \ \wedge\ p_i \models \varphi_i
\end{array}
$$

Figure 2. Satisfaction of formulae in the STS logic L_{PC}.

Definition 7 (STS Logic). *Let Σ_s be the set of operators in Σ which appear in the left-hand side of the conclusion of a proof rule of PC (e.g. the operators in $C[X_1,\ldots,X_n]$ for the rule of Definition 6). The* STS logic L_{PC} *associated to PC has as formulae*

$$\varphi ::= X \mid p \mid \diamond a.\varphi \mid f(\varphi,\ldots,\varphi)$$

where $X \in \mathcal{X}$, $p \in \mathcal{P}$, $a \in \Lambda$, $f \in \Sigma_s$.

A formula $f(\varphi_1,\ldots,\varphi_n)$ is satisfied by any component of the shape $f(p_1,\ldots,p_n)$ where each p_i satisfies φ_i. A formula $\diamond a.\varphi$ is satisfied by any component which is able to perform an a-labelled transition, evolving in a component satisfying φ. Since the logic will be used to label the transitions of an STS, according to the general assumptions in Section 2, process variables and (closed) components are included as atomic formulae. Observe that, if $\Sigma_s = \Sigma$ then all components can be inductively constructed as formulae of the kind $f(\varphi_1,\ldots,\varphi_n)$ with $f \in \Sigma$ and thus there is no need to add them explicitly (but in most PC no rule is given for the nil component 0, which is thus not in Σ_s). Satisfaction is formally defined in Fig. 2 (for any $p \in \mathcal{P}$ and for any formula φ in L_{PC}).

To understand the definition of the STS logic L_{PC} note that an instance $C[p_1,\ldots,p_n]$ of a given coordinator $C[X_1,\ldots,X_n]$, in order to perform a transition, must match the left-hand side of the conclusion of a rule. This might impose the components p_i's to have a certain structure, hence the need of inserting the spatial operators $f \in \Sigma_s$ in the logic. Furthermore, the premises of the matched rule must be satisfiable. Such premises usually require the components p_i's to be able to exhibit some behaviour, i.e. to perform a certain transition. Hence the logic includes also modal operators $\diamond a.(\_)$.

3.2 Algebraic PC without Structural Axioms

We next illustrate a constructive procedure for defining a correct and complete STS over the logic L_{PC} for a given process calculus PC whose proof rules are in algebraic format. Here we concentrate on process calculi *without* structural axioms. In Section 3.3 will discuss the refinements needed in the presence of structural axioms.

The STS over L_{PC} is specified by means of a Prolog program which can be used to compute the possible symbolic transitions of every coordinator.

Definition 8 (Prolog Program). *The Prolog program Prog*(PC) *associated to the process calculus PC contains as the first clause*

```
trs(box(A,X),A,X) :- !.
```

where `box` *is a new operator, not in Σ, and* `A` *is a variable that stands for any action. For any proof rule in PC of the shape outlined in Definition 6 also a clause*

```
trs(C[X1,...,Xn], a, D[Z1,...,Zn]) :- trs(Xi1,ai1,Yi1),...,trs(Xik,aik,Yik).
```

is included, where $\{i_1,\ldots,i_k\}$ *is the set of indexes* I *of the corresponding rule and* `Zi` *can be either* `Yi` *(when* $i \in I$*) or* `Xi` *(otherwise).*

The program $Prog(\mathsf{PC})$ defines the predicate `trs(X,A,Y)` whose intended meaning is "any component satisfying X can perform a transition labelled by A and become a component satisfying Y". Given a coordinator $C[X_1,\ldots,X_n]$, if the query

```
?- trs(C[X1,...,Xn], A, Z)
```

is successful, then the corresponding computed answer substitution can be seen as a symbolic step for the coordinator $C[X_1,\ldots,X_n]$: the computed answer substitutions for the variables `X1`, ..., `Xn` will represent the formulae in L_{PC} labelling the transition, `A` the action label and `Z` the target coordinator.

The first clause in $Prog(\mathsf{PC})$ can be unified only with a goal `trs(X,A,_)` whose first argument is a variable (since `box` is not an operator in PC). In this case there is no need of imposing structural requirements on X, since the only requirement for any component `X` for doing `a` and becoming `Y` is exactly `box(a,Y)`. Thus the goal is refuted just imposing a behavioural constraint on the component corresponding to `X`, i.e. by asking that `X` can perform an `A` action. The cut operator in the body of the clause avoids that subsequent refutations are tried, using different rules that could be otherwise matched by the goal `trs(X,A,_)`. To this aim, it is important that modal rules be listed first than all the other rules.

The second class of clauses in $Prog(\mathsf{PC})$ just represents a Prolog translation of the operational proof rules of the calculus. Each such clause imposes (by unification) the more general structural (spatial) constraints that the unspecified components of a coordinator should satisfy to allow the corresponding step. The requirements on the behaviour of the subcomponents, as expressed by the premises of the corresponding proof rule, are represented by the subgoals in the body of the clause.

The backtracking mechanism of Prolog and the use of meta-logic operators (like `bagof`) allow one to determine all the symbolic transitions for each coordinator C (finitely many under the assumption that the rules of the calculus and thus the program are finite). Hence the Prolog program $Prog(\mathsf{PC})$ can be seen as the specification of an STS for the process calculus PC over logic L_{PC}. The main result of this section states that such STS is correct and complete for the considered process calculus.

Theorem 3. *The* STS *specified by* $Prog(\mathsf{PC})$ *is correct and complete.*

Proof (Sketch). To prove correctness observe that if $C[X_1,\ldots,X_n] \xrightarrow{(\varphi_1,\ldots,\varphi_n)}_a D[Y_1,\ldots,Y_m]$ then there exists a refutation of the query

```
?- trs(C[X1, ..., Xn], a, Z)
```

with computed answer substitution `Xi` $=\varphi_i$ and `Z = D[Y1, ..., Yn]`. An inductive reasoning on the height of the refutation allows us to prove that for any $q_1,\ldots,q_m$ and $p_1,\ldots,p_n$ such that each $p_i \models \varphi_i[q_1/Y_1,\ldots,q_m/Y_m]$ there exists a derivation of $C[p_1,\ldots,p_n] \xrightarrow{a} D[q_1,\ldots,q_m]$.

```
trs( box(tau,X)  , tau ,   X  ) :- !.
trs(  a.X|'a     , tau , X ).
trs(    X|Y      , tau , X|Z ) :- trs(Y, tau, Z).
```

Figure 3. The Prolog program relative to the simple CCS-like calculus.

As for completeness, if $C[p_1,\dots,p_n] \xrightarrow{a} q$ then the corresponding derivation in the proof system of PC can be turned into a refutation witnessing that $C[X_1,\dots,X_n] \overset{(\varphi_1,\dots,\varphi_n)}{\longrightarrow}_a D[Y_1,\dots,Y_m]$. Furthermore $q \equiv D[q_1,\dots,q_m]$ and each $p_i \models \varphi_i[q_1/Y_1,\dots,q_m/Y_m]$. □

3.3 Algebraic PC with AC1 Parallel Composition Operator

To understand why the proposed approach must be extended to deal with structural axioms, we focus on a very common case, i.e., an algebraic PC with a *parallel composition operator* "|", subject to AC1 axioms (associativity, commutativity and identity)

$$(X \mid Y) \mid Z \equiv X \mid (Y \mid Z) \qquad X \mid Y \equiv Y \mid X \qquad X \mid \mathbf{0} \equiv X$$

where $\mathbf{0}$ is the inactive component. Furthermore we suppose that parallel composition allows a single component to move autonomously, performing an action that is reflected at topmost level, i.e., we assume that the proof rules for parallel composition include

$$\frac{X \xrightarrow{a} X'}{X \mid Y \xrightarrow{a} X' \mid Y}\ (par)$$

For the construction of the Prolog program *Prog*(PC) we first need to extend the set of proof rules of the calculus. Due to the presence of the associativity axiom, for any proof rule r of the calculus where "|" occurs in the left-hand side of the conclusion as topmost operator, we have to insert a new rule r'. The new rule is obtained from r by adding in parallel a generic idle component, i.e. for any rule of the kind

$$\frac{\{X_i \xrightarrow{a_i} Y_i\}_{i\in I}}{C_1[X_1,\dots,X_n] \mid C_2[X_{n+1},\dots,X_{n+m}] \xrightarrow{a} D[Z_1,\dots,Z_{n+m}]}$$

we add a new rule (analogous to the completion in rewriting systems modulo AC1)

$$\frac{\{X_i \xrightarrow{a_i} Y_i\}_{i\in I}}{C_1[X_1,\dots,X_n] \mid C_2[X_{n+1},\dots,X_{n+m}] \mid X_{n+m+1} \xrightarrow{a} D[Z_1,\dots,Z_{n+m}] \mid X_{n+m+1}}$$

Then *Prog*(PC) is defined exactly as before. Of course unification must be considered up to AC1 structural axioms (see algorithms and further references in [18,4]).

Example 2. Consider a simple CCS-like calculus, with AC1 parallel composition and only one rule for asynchronous communication

$$\frac{}{a.X \mid \bar{a} \xrightarrow{\tau} X}$$

The Prolog program induced by the original proof rule is shown in Fig. 3, where `'a` is the program representation for action $\bar{a}$. The following query

```
?- trs(a.0|a.0|X, A, Z)
```

would return the substitutions `X = 'a` for `X` and `Z = a.0` for `Z`; but `X = 'a` is not the more general substitution for X that allows the context to perform the step. In fact, the coordinator $a.0|a.0|X$, instantiated with a component p satisfying the formula $\varphi = \bar{a}$ returned by the Prolog program (namely with $p = \bar{a}$), could perform only one step, but, obviously, X could also be instantiated with the component $\bar{a} \mid \bar{a}$, allowing the coordinator to perform two steps. Actually, `X = 'a | Y` results to be the more general substitution which, thanks to the identity axiom, "comprises" the previous one. In order to obtain such a computed answer from the program, it is enough to extend the proof system with the rule r'

$$\frac{}{a.X \mid \bar{a} \mid Y \xrightarrow{\tau} X \mid Y}$$ □

It is easy to show that the new proof rules r' are valid in the original proof system, hence the extension of the proof system does not change the semantics of the PC. Due to the presence of the identity axiom, for any r' we can also remove the original rule r without affecting the semantics of the calculus. The result expressed by Theorem 3 extends also to this case, i.e., the STS specified by $Prog(\mathsf{PC})$ is correct and complete.

An analogous approach can be followed to deal with a *replication* operator "!", subject to the structural axiom $!X \equiv !X \mid X$.

4 Case Study: A Basic Calculus for Mobility

We consider a basic calculus for mobility (BCM) which can be seen as an asynchronous version of CCS [21], enriched with ambients, or, alternatively, as (a restriction-free version of) the ambient calculus [12] with asynchronous CCS-like communication.

Definition 9 (BCM). *Let* $\mathbf{A}$ *be a set of channels and let* $\mathcal{N}$ *be a set of ambient names. The set of* BCM *processes* $\mathcal{P}$ *is defined by the grammar:*

$$P \ ::= \ 0 \mid \bar{a} \mid a.P \mid open\, n.P \mid in\, n.P \mid out\, n.P \mid n[P] \mid P|P$$

with $a \in \mathbf{A}$*,* $n \in \mathcal{N}$*, and where the parallel operator is AC1:*

$$P|(Q|R) \equiv (P|Q)|R \qquad P|Q \equiv Q|P \qquad P|0 \equiv P$$

The operational semantics of BCM is defined by the SOS operational rules in Fig. 4. The rules *open*, *in*, and *out* are the classical rules of ambient calculus; communication (rule *com*) is allowed only inside the same ambient; reductions can happen under any ambient and in any parallel process (but not under prefixes), as stated by rules *amb* and *par*, respectively. Since the semantics is presented as a reduction system, transitions have no label (or equivalently they can be thought of as having all the same label τ).

The logic $\mathsf{L}_{\mathrm{BCM}}$ over the set of components is defined as explained in Section 3.1. The set of spatial operators of the logic includes all the operators of the signature, i.e., $\Sigma_s = \Sigma$, so as to characterise all the possible transitions of the semantics of BCM (strictly speaking, $0 \notin \Sigma_s$, but its presence is harmless and makes the notation simpler). All axioms in Fig. 4 introduce the need of spatial formulae (the lefthand side of the reduction

$$\frac{}{n[P] \mid open\ n.Q \ \rightarrow\ P|Q}\ (open) \qquad \frac{}{n[P]|m[in\ n.Q|R] \ \rightarrow\ n[P|m[Q|R]]}\ (in)$$

$$\frac{}{n[P|m[out\ n.Q|R]] \ \rightarrow\ n[P]|m[Q|R]}\ (out) \qquad \frac{}{n[a.P|\bar{a}|Q] \ \rightarrow\ n[P|Q]}\ (comm)$$

$$\frac{P \ \rightarrow\ Q}{n[P] \ \rightarrow\ n[Q]}\ (amb) \qquad \frac{P \ \rightarrow\ Q}{P|R \ \rightarrow\ Q|R}\ (par)$$

Figure 4. Operational semantics of BCM.

requires a specific structure of the component). The rules *amb* and *par*, instead, calls for modal formulae, since their premises refer to observable behaviours and not to the structure of the components. The formulae φ of the logic $\mathsf{L}_{\mathrm{BCM}}$ are:

$$\varphi ::= X \mid \diamond.\varphi \mid 0 \mid \alpha.\varphi \mid n[\varphi] \mid \varphi_1|\varphi_2,$$

where $X \in \mathcal{X}$, $n \in \mathcal{N}$ and $\alpha \in \{a, \bar{a}, open\ n, in\ n, out\ n\}$. Since transitions are not labelled, the modal operator does not refer to any action. The notion of satisfaction for $\mathsf{L}_{\mathrm{BCM}}$ ($P \models \varphi$) is defined like in the general case (see Fig. 2). Then we can consider the correct and complete STS for BCM specified by the Prolog program *Prog*(BCM).

To have a grasp of the properties of the calculus, let us consider two ambients with different names $n[a.0 \mid \bar{a}.0]$ and $m[b.0 \mid \bar{b}.0]$. Both processes are able to perform an internal communication according to rule *comm*, evolving to a (deadlocked) ambient containing the nil component 0. Straightforwardly,

$$n[a.0 \mid \bar{a}.0] \sim m[b.0 \mid \bar{b}.0],$$

i.e. internal actions do not distinguish ambients. It is easy to show that bisimilarity is not a congruence for this calculus, since the above bisimilar processes are distinguished when put in parallel with *open* $n.0$ (it interacts with $n[a.0 \mid \bar{a}.0]$ but not with $m[b.0 \mid \bar{b}.0]$).

Processes $n[a.0 \mid \bar{a}.0]$ and $m[b.0 \mid \bar{b}.0]$ are (bisimilar) instances of the coordinators $n[X]$ and $m[X]$. It is easy to verify $n[X] \not\sim_{\mathrm{strict}} m[X]$, in fact, due to rule *out*:

$$n[X] \ \stackrel{Y|m[out\ n.\ Z|W]}{\longrightarrow}\ n[Y] \mid m[Z \mid W],$$

while $m[X]$ has an analogous transition but with a different label and conclusion:

$$m[X] \ \stackrel{Y|n[out\ m.\ Z|W]}{\longrightarrow}\ m[Y] \mid n[Z \mid W].$$

Actually, $n[X] \not\sim m[X]$, since they are distinguished by $X = k[out\ n.0]$, and hence, by Theorem 2, $n[X] \not\sim_{\mathrm{large}} m[X]$. An example of coordinators related by $\sim_{\mathrm{strict}}$, and hence, using Theorems 1 and 2, also by $\sim_{\mathrm{large}}$ and $\sim$, is: $n[m[out\ n.X]] \sim_{\mathrm{strict}} n[0] \mid m[a \mid \bar{a}.X]$. In fact, the two coordinators have the only symbolic transitions below, which lead to obviously bisimilar coordinators:

$$n[m[out\ n.X]] \ \stackrel{Y}{\longrightarrow}\ n[0] \mid m[Y] \quad \text{and} \quad n[0] \mid m[a \mid \bar{a}.X] \ \stackrel{Y}{\longrightarrow}\ n[0] \mid m[Y].$$

Conclusions

We have illustrated a general methodology for reasoning about open systems, viewed as coordinators in suitable process calculi, with special interest in bisimilarity. For a PC and a process logic which characterises the structural and behavioural properties of interest, we have introduced a notion of (correct and complete) symbolic transition system, where states are coordinators and transitions are labelled by logic formulae expressing the requirements which uninstantiated components should satisfy for the transition to happen. Over an STS two symbolic bisimilarities can be defined, the *strict bisimilarity* and the "coarser" *large bisimilarity*, both refining the universal bisimilarity on coordinators which takes all possible closed instantiations. For algebraic PC, whose rules are in a quite general format, we have also provided a constructive way of deriving a spatio-temporal logic and a (correct and complete) symbolic transition system over such logic. The applicability of the proposed methodology has been finally illustrated by means of a toy process calculus with CCS-like communication within ambients.

An interesting issue which has not been faced here is the treatment of names and name restriction, which plays a basic role in the specification of systems with fresh or secret resources. While the notions of (correct and complete) STS and the results about symbolic bisimulation are parametric w.r.t. the chosen process logic, the constructive definition of the correct and complete STS for a given process calculus, presented in Section 3, and especially the definition of the underlying process logic, should be extended to deal with a logical notion of freshness. A source of inspiration could be the work of Cardelli and Caires [9,10].

We already mentioned that symbolic transitions have been studied by several authors, e.g. Sewell [25], and Leifer and Milner [20] in order to avoid universal quantification over contexts. These approaches, where steps can perform contextual closures, can be seen as the dual of our approach, where steps can instantiate contexts. It would be interesting to give a formal account of this duality, and, in particular, to see if the categorical approach of [20], based on relative pushouts, can be dualised in our case resorting to a notion of relative pullback.

Recently, the symbolic approach to the verification of infinite state cryptographic protocols has attracted a lot of interest. Some authors use logic abstractions to characterise symbolic states [1,13], others exploit, in particular, the generality of unification to devise minimal assumptions over symbolic states [5]. Pursuing further the similarities of our symbolic approach to bisimulation with these approaches, so as to apply our methodology to the field, appears to be a stimulating line of future research.

Regarding the automatic construction of STS, we plan to generalise it to *meta* and *abductive* Logic Programming. The first one should allow for the programmable definition of proofs, and hence for more specific reasoning over the structure of a PC. The second one should provide the means for hypothetical (assumption-based) reasoning about the properties labelling STS, allowing to answer questions like "under which circumstances (assumptions) the process $P \mid X$ can evolve so as to satisfy a given property?", typically relevant in open and dynamic system engineering [3,2].

Acknowledgements. We would like to thank Narciso Martí-Oliet, Sabina Rossi and the anonymous referees for their helpful comments and suggestions.

References

1. M. Abadi and M.P. Fiore. Computing symbolic models for verifying cryptographic protocols. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pp. 160–173. IEEE, 2001.
2. R. Allen and D.Garlan. A Formal Basis for Architectural Connectors. *ACM TOSEM*, 6(3), pp. 213–249, 1997.
3. L.F. Andrade, J.L. Fiadeiro, J. Gouveia, G. Koutsoukos and M.Wermelinger. Coordination for Orchestration. In *Coordination Models and Languages,* 5th Int. Conference COORDINATION. *Lect. Notes in Comput. Sci.* 2315 pp. 5–13. Springer 2002.
4. F. Baader and W. Snyder. Unification theory. In *Handbook of Automated Reasoning*. Elsevier Science, 2000.
5. M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proc. ICALP'01*, *Lect. Notes in Comput. Sci.* 2076, pp. 667–681. Springer, 2001.
6. R. Bruni, D. de Frutos-Escrig, N. Martí-Oliet, and U. Montanari. Bisimilarity congruences for open terms and term graphs via tile logic. In *Proc. CONCUR 2000*, *Lect. Notes in Comput. Sci.* 1877, pp. 259–274. Springer, 2000.
7. R. Bruni, U. Montanari, and F. Rossi. An interactive semantics of logic programming. *Theory and Practice of Logic Programming*, 1(6):647–690, 2001.
8. L. Caires. *A Model for Declarative Programming and Specification with Concurrency and Mobility*. PhD thesis, Departamento de Informática, Universidade Nova de Lisboa, 1999.
9. L. Caires and L. Cardelli. A spatial logic for concurrency (part I). In *Proc. TACS 2001*, *Lect. Notes in Comput. Sci.* 2215, pp. 1–37. Springer, 2001.
10. L. Caires and L. Cardelli. A spatial logic for concurrency (part II). In *Proc. CONCUR 2002*. *Lect. Notes in Comput. Sci.*, Springer, 2002. To appear.
11. L. Cardelli and A.D. Gordon. Anytime, anywhere. modal logics for mobile ambients. In *Proc. POPL 2000*, pp. 365–377. ACM, 2000.
12. L. Cardelli and A.D. Gordon. Mobile ambients. In *Proc. FoSSaCS'98*, *Lect. Notes in Comput. Sci.* 1378, pp. 140–155. Springer, 1998.
13. E.M. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. PROCOMET'98*. Chapmann & Hall, 1998.
14. R. De Simone. Higher level synchronizing devices in MEIJE-SCCS. *TCS*, 37:245–267, 1985.
15. J.L. Fiadeiro, T. Maibaum, N. Martí-Oliet, J. Meseguer, and I. Pita. Towards a verification logic for rewriting logic. In *Proc. WADT'99*, *LNCS* 1827, pp. 438–458. Springer, 2000.
16. F. Gadducci and U. Montanari. The tile model. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.
17. M. Hennessy and H. Lin. Symbolic bisimulations. *Theoret. Comp. Sci.*, 138:353–389, 1995.
18. A. Herold and J. Siekmann. Unification in abelian semi-groups. *Journal of Automated Reasoning*, 3(3):247–283, 1987.
19. K.G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. In *Proc. ICALP'90*, *Lect. Notes in Comput. Sci.* 443, pp. 526–539. Springer, 1990.
20. J.J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In *Proc. CONCUR 2000*, *Lect. Notes in Comput. Sci.* 1877, pp. 243–258. Springer, 2000.
21. R. Milner. *A Calculus of Communicating Systems*, *LNCS* 92. Springer, 1980.
22. U. Montanari and V. Sassone. Dynamic congruence vs. progressing bisimulation for CCS. *Fundamenta Informaticae*, 16:171–196, 1992.
23. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Computer Science Department, 1981.
24. A. Rensink. Bisimilarity of open terms. *Inform. and Comput.*, 156(1-2):345–385, 2000.
25. P. Sewell. From rewrite rules to bisimulation congruences. In *Proc. CONCUR'98*, *Lect. Notes in Comput. Sci.* 1466, pp. 269–284. Springer, 1998.