A Semantic Framework for Open Processes

P. Baldan(1), A. Bracciali(2), R. Bruni(2)

(1) Dipartimento di Matematica Pura e Applicata,
Università di Padova (Italy)
baldan@math.unipd.it

(2) Dipartimento di Informatica,
Università di Pisa (Italy)
{bruni,braccia}@di.unipi.it

Abstract

We propose a general methodology for analysing the behaviour of open systems modelled as *coordinators*, i.e., open terms of suitable process calculi. A coordinator is understood as a process with holes or place-holders where other coordinators and components (i.e., closed terms) can be plugged in, thus influencing its behaviour. The operational semantics of coordinators is given by means of a symbolic transition system, where states are coordinators and transitions are labelled by spatial/modal formulae expressing the potential interaction that plugged components may enable. Behavioural equivalences for coordinators, like strong and weak bisimilarities, can be straightforwardly defined over such a transition system. Differently from other approaches based on universal closures, i.e., where two coordinators are considered equivalent when all their closed instances are equivalent, our semantics preserves the openness of the system during its evolution, thus allowing dynamic instantiation to be accounted for in the semantics. To further support the adequacy of the construction, we show that our symbolic equivalences provide correct approximations of their universally closed counterparts, coinciding with them over closed components. For process calculi in suitable formats, we show how tractable symbolic semantics can be defined constructively using unification.

1 Introduction

In mobile and distributed programming, *interaction* and *coordination* have become as much important issues as computation, and large systems have become environments more and more *open* to the dynamic connection with agents, components and services. Given the high complexity of such open systems, formal models are needed to drive their specification, design, analysis and verification for a successful deployment.

Process calculi are a quite popular formal model, often instrumental in focusing on certain key aspects like communication and distribution. Depending on the desired abstraction level, the openness of the modelled systems can be expressed in many different ways. For example, in the π-calculus, processes can have free channels on which fresh channels can be extruded to establish dynamic links for communication and thus an open system is just a π-process with some free channels. However, this vision, that we can call *name-driven*, can blur to some extent the conceptual distinction between *components* (computational entities) and *coordinators* (open interaction environments). Here we take a different perspective, that we call *variable-driven*, which appears to be rather natural and that can be applied also to calculi simpler than the π-calculus, like CCS and other non-nominal calculi. It relies on the obvious distinction between contexts $C[X_1, ..., X_n]$, namely terms over the signature Σ of the calculus with process variables $X_1, \ldots, X_n$, and ground terms p, namely terms with no process variables. The former are natural candidates for coordinators, with process variables denoting the holes where processes can be plugged in, while the latter are the natural candidates for components.

The operational semantics and the associated behavioural equivalences (e.g., bisimilarities, traces, testing) are often defined for components. A satisfactory definition of the semantics of coordinators is not straightforward. A natural possibility consists of lifting the equivalences over components to coordinators by universal closure, in an extensional style: $C[X_1, ..., X_n]$ and $D[X_1, ..., X_n]$ are considered as equivalent if $C[p_1, ..., p_n]$ and $D[p_1, ..., p_n]$ are so for all components $p_1, ..., p_n$ that can be plugged in (as done, e.g., [43]).

In this paper we promote a symbolic approach to the semantics of coordinators by defining the operational semantics directly for coordinators. The rationale for doing this is motivated by the following arguments:

- *Open systems can evolve.* In the context of environments open to multi-party interaction, it may be of interest being able to study the behaviour of components inserted in partially instantiated coordinators, i.e., in coordinators where some holes are still available for being later instantiated. Hence the capability of formalising the evolution of general, partially instantiated coordinators is a primary issue. With respect to the aforementioned extensional semantics based on universal closure, where the semantics is evaluated by fixing from the beginning the whole scenario of execution, an explicit definition of the semantics of coordinators allows a higher level of dynamics, which is typical of open systems, to be accounted for. For instance, different components may join at different times and the choice of the next component to plug in can be influenced by the components plugged in so far.

Since components can be regarded as a special kind of coordinators, it is worth remarking that, on components, all equivalences arising from our constructions coincide with the corresponding standard behavioural equivalences for closed systems. Additionally, on coordinators, symbolic equivalences are finer than (or equal to) the corresponding equivalences obtained by universal closure.

- *Constructive methodology.* The need for the formal verification of open systems calls for the support of automated tools. Our abstract theory is complemented with an unification-based constructive counterpart that supports the automation of the basic steps of the methodology, providing the basis for the development of more sophisticated tools (initial experiments in this sense have already been carried out). The use of most general unifiers for inferring the transitions of coordinators guarantees that branching is kept to the necessary least.

- *Complementarity with contextualisation techniques.* Sewell's paper [43] opened a flourishing research thread on finding methodologies for the synthesis of labelled transition systems starting from reduction systems. The idea is to use contexts as labels and to derive labelled transition systems for which bisimilarity is a congruence. The transitions are limited to those contexts that can play an active part in the reduction of the source state. The congruence property is of course an important achievement, as it allows modular reasoning. The states of the synthesised labelled transition system are ground, but transitions can lead to terms with variables (i.e., contexts), so that bisimilarity must be extended to contexts and this is done by considering all possible closed instances. Hence the problem of not exposing all contexts in the labels is somehow shifted to exposing all the possible instances. Our framework is precisely targeted to find/characterise an alternative abstract equivalence over contexts, so that, as explained in more detail in Section 7, our approach can be considered to some extent complementary to the one in [43].

- *Generality and friendly notation.* Our approach generalises previous different variable-driven approaches in the literature (like [25, 38], see Section 7 for more details) allowing us to deal with more expressive observations, which capture both the behaviour and the structure of the plugged in components. Moreover, the framework is largely independent of the underlying calculus. On the other hand, we have done at our best to keep the notation easy to grasp, with the aim of making it widely accessible.

The semantics we propose for coordinators is given in terms of a peculiar transition system. Its main distinguishing feature is the idea of labelling the transitions exiting from coordinators with trigger-effect pairs $\langle\varphi, a\rangle$. The trigger φ is intended to provide an abstract characterisation of the structure that a component must possess and of the actions it can perform in order to allow a transition with label a to fire. The choice of mixing structural and behavioural constraints can be understood by recalling that the operational semantics of process calculi can often be conveniently expressed by a set of inductive proof rules which, according to Plotkin's SOS seminal paper [37], define the behaviour of each component on the basis of two elements: the structure of the component and the behaviour of its subcomponents. Intuitively, a coordinator can either evolve autonomously or interact with the components which are plugged in and, by the above considerations, the interaction can depend:

1. on the spatial structure of the components that are inserted in / connected with the coordinator;
2. on the observable behaviour such components can exhibit, i.e. on the transitions that they can perform.

We propose to use as triggers formulae from a suitable logic for expressing the most general class of components with whom the coordinator can perform a specific transition. Since a single formula can characterise potentially infinite classes of components, this approach can drastically reduce the size of the transition system. More specifically, we introduce a notion of *symbolic transition system* (STS for short), whose states are coordinators and whose transitions have the shape

$$C[X_1, ..., X_n] \overset{\varphi_1,...,\varphi_n}{\longrightarrow}_a D[Y_1, \ldots, Y_m]$$

meaning that a transition labelled with a can be performed by $C[p_1, ..., p_n]$ whenever each component p_i satisfies the corresponding formula φ_i, and the target state will be a suitable instance of $D[Y_1, \ldots, Y_m]$. The logic where the formulae φ_i's live and the notion of satisfaction must be of course targeted to the process calculus under study. As suggested above, in general the logic may involve both spatial and behavioural aspects of components, i.e. it can be a *spatial logic* along the lines of [14, 18].

For process calculi whose operational semantics is defined by inference rules in quite general SOS formats, e.g., the algebraic [25] or the positive GSOS format [8] (both including also the popular De Simone format [22]), we give an algorithm, expressed as a Prolog program, for building a sound and complete symbolic transition system for the calculus. The use of unification guarantees a minimality property for the generated transition systems.

Given a symbolic transition system for a process calculus several behavioural equivalences can be defined directly for coordinators. In this paper we focus on bisimilarities, which are by far the most popular equivalences, but the theory is general enough to encompass several other behavioural equivalences (e.g., those based on traces as shown in [7]).

Two kinds of bisimilarities $\sim_{\mathrm{s}}$ and $\dot{\sim}_{\mathrm{l}}$, called respectively *strict* and *loose* bisimilarities (introduced as "*large* bisimilarity" in [5]) are defined on coordinators. Strict bisimilarity $\sim_{\mathrm{s}}$ is a straight extension of the standard bisimilarity on labelled transition systems. Loose bisimilarity $\dot{\sim}_{\mathrm{l}}$, is obtained by relaxing the requirements when comparing labels in the bisimulation game: roughly, a transition $C[X] \xrightarrow{\varphi}_a C'[X]$ can be simulated by a transition $D[X] \xrightarrow{\psi}_a D'[X]$, where ψ imposes weaker structural requirements than φ. For sound and complete STSs both "symbolic" bisimilarities imply $\sim_{\mathrm{u}}$, the standard lifting of bisimilarity $\sim$ to coordinators defined by universal closure. More precisely, loose bisimilarity $\dot{\sim}_{\mathrm{l}}$ approximates $\sim_{\mathrm{u}}$ better than $\sim_{\mathrm{s}}$, although in general $\dot{\sim}_{\mathrm{l}}$ is non-transitive (incidentally, the dot on top of $\dot{\sim}_{\mathrm{l}}$ is a reminder of this fact).

Many process calculi include in the set of labels a special *silent* action, which models internal (non-observable) computations. For such calculi, (strong) bisimilarity can be too fine, distinguishing components on the basis of internal computations that do not require any observable interaction. In such cases weak bisimilarities, which "ignore" silent actions, are often considered more appropriate. We show that our approach naturally extends to weak behavioural equivalences. More precisely, we define a symbolic counterpart of weak bisimilarity, called *weak symbolic bisimilarity* $\approx_{\mathrm{w}}$, and we show that, as for strong equivalences, it implies the corresponding equivalence, obtained by lifting the relation over components by universal closure.

Structure of the paper. In Section 2 we fix the notation and we give some basic definitions. In Section 3 we overview the general ideas on which our approach relies, introducing the notion of (sound and complete) symbolic transition system. In Section 4 we define strong and weak symbolic bisimilarities, showing that each symbolic bisimilarity is finer than the corresponding equivalence based on universal closure. In Section 5, we illustrate the algorithmic construction of the symbolic transition system for process calculi with operational rules in the algebraic format, we prove its correctness and we present a minimality result enjoyed by the construction. In Section 6, in order to illustrate the applicability of the approach, we show how it can be adapted in the presence of structural axioms and for calculi in positive GSOS format. Minimality properties enjoyed by these constructions are also discussed. In Section 7, we report on some related literature, while in Section 8 we draw some conclusions and directions for further research. Some minor proofs have been omitted or only sketched and can be found in [6]. A preliminary version of this paper has been presented as [5].

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

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

Figure 1: Closure of structural axioms.

2 Background

2.1 Notation

To ease the presentation we will consider only unsorted signatures, though our results can be extended 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) and we let $\mathbb{T}_\Sigma = \mathbb{T}_\Sigma(\emptyset)$. For $P \in \mathbb{T}_\Sigma(\mathcal{X})$ we denote by $var(P)$ the set of all variables $X \in \mathcal{X}$ that appear in P. If $var(P) = \emptyset$ then P is called *closed*, otherwise *open*. A term is *linear* if each variable occurs at most once in it.

When signatures are used to present the syntax of process calculi, closed terms define the set $\mathcal{P}$ of *components* of the calculus, ranged over by p, q, ..., while the general, possibly open, terms form the set $\mathcal{C}$ of *coordinators*, ranged over by C, D, ... We shall write $C[X_1, \ldots, X_n]$, often abbreviated $C[\vec{X}]$ to mean that C is a coordinator such that $var(C) \subseteq \{X_1, \ldots, X_n\}$. Unless stated otherwise, all coordinators considered in the paper will be assumed to be *linear*.

Given a tuple $\vec{p} = p_1, \ldots, p_n$ of components, we denote $C[p_1, \ldots, p_n]$ (or $C[\vec{p}]$ for short) the component $C[p_1/X_1, \ldots, p_n/X_n]$ obtained from $C[\vec{X}]$ by replacing each occurrence of variable X_i with p_i for $i \in \{1, \ldots, n\}$.

A *structural axiom* is a sentence of the form $P \equiv Q$ for $P, Q \in \mathbb{T}_\Sigma(\mathcal{X})$. Let $E = \{P_i \equiv Q_i \mid i \in I\}$ be a set of structural axioms. The initial algebra $\mathbb{T}_{\Sigma,E}$ is the quotient of $\mathbb{T}_\Sigma$ modulo the equivalence $\equiv$ defined in Figure 1, where axioms in E are closed w.r.t. substitution, contextualisation, reflexivity, symmetry, and transitivity.

2.2 Strong and weak bisimulation for process calculi

Process calculi come often equipped with an operational semantics which consists of a *labelled transition system* (LTS) over an alphabet of labels Λ, where states are components over a process signature Σ and transitions model basic activities of the system. More precisely the LTS will consist of a pair $\langle \mathcal{P}, \to \rangle$ where $\mathcal{P}$ is the set of components and $\to \subseteq \mathcal{P} \times \Lambda \times \mathcal{P}$ is the transition relation, whose elements (p, a, q) are usually written as $p \to_a q$. Commonly such LTS is specified by means of a collection of inductive proof rules of the kind

$$\frac{q_1 \to_{a_1} q_1' \qquad \ldots \qquad q_n \to_{a_n} q_n'}{p \to_a p'}$$

which axiomatise the relation $\to$ over components. In the presence of structural axioms E, proof rules typically include:

$$P \ ::= \ 0 \mid \alpha.P \mid open\ n.P \mid in\ n.P \mid out\ n.P \mid n[P] \mid P \mid P$$

$$\frac{}{\alpha.P \ \rightarrow_\alpha \ P}\ (pref) \qquad \frac{P \ \rightarrow_\ell \ Q}{P \mid R \ \rightarrow_\ell \ Q \mid R}\ (par)$$

$$\frac{P_1 \ \rightarrow_\alpha \ Q_1 \quad P_2 \ \rightarrow_{\bar{\alpha}} \ Q_2}{P_1 \mid P_2 \ \rightarrow_\tau \ Q_1 \mid Q_2}\ (comm)$$

$$\frac{}{n[P] \mid open\ n.Q \ \rightarrow_\tau \ P \mid Q}\ (open) \qquad \frac{P \ \rightarrow_\tau \ Q}{n[P] \ \rightarrow_\tau \ n[Q]}\ (amb)$$

$$\frac{}{n[P] \ \mid \ m[in\ n.Q \mid R] \ \rightarrow_\tau \ n[P \mid m[Q \mid R]]}\ (in)$$

$$\frac{}{n[P \mid m[out\ n.Q \mid R]] \ \rightarrow_\tau \ n[P] \mid m[Q \mid R]}\ (out)$$

Figure 2: Syntax and operational semantics of ACCS.

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

i.e., states are essentially equivalence classes $[p]_\equiv$ of components (modulo $\equiv$).

In the following, we assume that a process calculus PC is fixed with signature Σ and structural axioms E, whose semantics is given by an LTS $\mathcal{L}$ over $\mathbb{T}_{\Sigma,E}$ and label alphabet Λ.

Example 1 An example of a process calculus for mobility, which will be often referred later, is the ACCS calculus, which can be seen as (a restriction-free version of) the ambient calculus [17] with CCS-like communication [35]. Let $\mathbf{A}$ be a set of channels and let $\mathcal{N}$ be a set of ambient names. Then, ACCS is defined by the grammar and SOS operational rules in Figure 2, where $\alpha \in \mathbf{Act} = \{a, \bar{a} : a \in \mathbf{A}\}$ and $n \in \mathcal{N}$. The parallel operator is AC1, i.e., associative, commutative and with the nil process 0 as the unit. The rules *pref*, *com* and *par* are the usual CCS rules (where it is intended that $\bar{\bar{\alpha}} = \alpha$, $\tau \notin \mathbf{Act}$ is a distinguished action, and $\ell \in \mathbf{Act} \cup \{\tau\}$). The rules *open*, *in*, and *out* are the classical rules of ambient calculus. By rule *amb*, communication is only allowed within the same ambient. $\square$

Several notion of observational equivalences can be defined on top of an LTS. We focus on bisimilarities, which are by far the most popular equivalences.

Definition 2 (Strong bisimilarity $\sim$) *A* strong bisimulation *is a symmetric relation $\div$ over components such that if $p \div q$, then for any transition $p \rightarrow_a p'$ a component q' and a transition $q \rightarrow_a q'$ exist such that $p' \div q'$. We denote by $\sim$ the largest bisimulation and call it* strong bisimilarity *or just* bisimilarity*.*

Note that the presence of rule (*equiv*) guarantees that for any p and q we have that if $p \equiv q$ then $p \sim q$.

In many situations the set of labels Λ contains a special *silent* action τ, which models internal (non-observable) computations. In such cases, strong bisimilarity can be too fine, distinguishing components whose behaviours differ only in the length of some internal computations that do not require any observable interaction. A more suitable, relaxed notion of behavioural equivalence is then provided by *weak bisimilarity*, which is coarser than $\sim$.

Definition 3 (Weak transitions) *Given a transition system over components, whose transition relation $\rightarrow$ can include silent steps $\rightarrow_\tau$, we define:*

- $p \Rightarrow_a q$ *when* $p(\rightarrow_\tau)^* \rightarrow_a (\rightarrow_\tau)^* q$, *and*
- $p \Rightarrow q$ *when* $p(\rightarrow_\tau)^* q$.

Definition 4 (Weak bisimilarity $\approx$) *A symmetric relation $\div$ over components is a* weak bisimulation *if for any $p \div q$*

1. *if $p \rightarrow_a p'$ then $q \Rightarrow_a q'$ and $p' \div q'$*
2. *if $p \rightarrow_\tau p'$ then $q \Rightarrow q'$ and $p' \div q'$*

We denote by $\approx$ the largest weak bisimulation and call it weak bisimilarity.

A natural way of lifting strong and weak bisimilarities (as well as any other behavioural equivalence defined on components) to coordinators consists of considering all possible closed instances of the coordinators.

Definition 5 (Universal closure) *Let $\div$ be an equivalence relation defined on components. The* universal closure *of $\div$ is the equivalence relation $\div_{\mathrm{u}}$ on coordinators defined by letting*

$$C[X_1, ..., X_n] \div_{\mathrm{u}} D[X_1, ..., X_n] \stackrel{\text{def}}{\Longleftrightarrow} \forall p_1, ..., p_n \in \mathcal{P}, \quad C[p_1, ..., p_n] \div D[p_1, ..., p_n]$$

In particular $\sim_{\mathrm{u}}$ and $\approx_{\mathrm{u}}$ will denote the universal closures of $\sim$ and $\approx$, respectively.

2.3 A spatial logic

As discussed in the introduction, we shall introduce a special kind of transition systems whose transitions are labelled with logic formulae that characterise suitable classes of components that play active roles in the transitions. The logic providing the labels is built over a fixed logic over components. For the purposes of this paper we consider $\mathsf{L_{PC}}$ which has *modal* and *spatial* operators in the style of [14, 18].

It is worth observing that the word “spatial” has been used in the literature to refer to the logical or physical distribution of system components, e.g., prefix in CCS is generally not taken as a spatial operator. For the aim of this paper, this word refers to the structure of a term and any operator of the signature can be, in principle, considered spatial. The syntax of (ground) $\mathsf{L_{PC}}$-formulae φ and the associated notion of satisfaction are given below.

Definition 6 (Ground formulae) *The set Φ of* (ground) $\mathsf{L_{PC}}$-formulae *φ is defined by the grammar*

$$\varphi ::= \top \mid f(\varphi, \ldots, \varphi) \mid \diamond a\, \varphi$$

where $f \in \Sigma$ is an operator in the process signature and $a \in \Lambda$ an action label.

The formula $\top$ (true) is satisfied by any component. The formula $f(\varphi_1, \ldots, \varphi_n)$ is satisfied by any component p that can be decomposed as $p \equiv f(p_1, \ldots, p_n)$ such that each p_i satisfies the corresponding formula φ_i. The formula $\diamond a\, \varphi$ is satisfied by any process p that can perform a transition labelled with a ending in a state q that satisfies φ. Formally, the satisfaction relation is defined as follow.

Definition 7 (Satisfaction) *The* satisfaction relation *is the least relation* $\_ \models \_ \subseteq \mathcal{P} \times \Phi$ *such that*

$$\begin{array}{lll} p \models \top & & \\ p \models f(\varphi_1, \ldots, \varphi_n) & \textit{if} & \exists p_1, \ldots, p_n.\ p \equiv f(p_1, \ldots, p_n) \ \wedge\ \forall i.\ p_i \models \varphi_i \\ p \models \diamond a\, \varphi & \textit{if} & \exists q.\ p \rightarrow_a q \ \wedge\ q \models \varphi \end{array}$$

We say that a component p satisfies *the formula φ, if $p \models \varphi$.*

A formula in $\mathsf{L_{PC}}$ is called *purely spatial* if it does not contain the modal operator $\diamond a\, \_$. Thus, abusing the notation, each component q can also be regarded as a (purely spatial) formula and it can be readily proved that

$$p \models q \quad \text{iff} \quad p \equiv q.$$

3 An operational semantics for coordinators

As we argued in the introduction, from a conceptual point of view, resorting to the universal closure is not the only natural way of providing a notion of equivalence for coordinators. In fact, a coordinator can be intended as an open system whose unspecified subcomponents can be progressively instantiated during the computation, while considering "universal" equivalences amounts to fixing the closed instances of the compared coordinators once and for all at the beginning of the computation. Moreover, from the point of view of automatic reasoning, the definition of behavioural equivalences over coordinators based on the closure with respect to any possible substitution presents some drawbacks. In fact, to verify the equivalence of two coordinators, one is typically led to check the equivalence of infinitely many processes (all the possible closed instances of the coordinators).

For the above reasons we have found convenient to define operational and abstract semantics of coordinators, directly, by exploiting a symbolic approach based on the following principles:

1. abstracting from components not playing an active role in the transition;
2. specifying the active components as little as possible;
3. making assumptions both on the structure and 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 that components should fulfil for enabling the move.

3.1 Formulae as labels

We first define the logic whose formulae will be used as labels for the transitions of coordinators. The logic must be powerful enough to be able to express, for any coordinator, the structural and behavioural properties which should be fulfilled by unspecified components to allow transitions to happen. A natural choice for such logic is a slight extension of the spatial logic L_{PC} over components, but, as discussed in [5], the approach is actually parametric with respect to the process logic over components one starts from.

Definition 8 (sts logic) *The* sts logic $\mathsf{SL}_{\mathsf{PC}}$ *associated to* PC *has as formulae*

$$\varphi ::= X \mid \diamond a\, \varphi \mid f(\varphi, \dots, \varphi)$$

where $X \in \mathcal{X}$, $p \in \mathcal{P}$, $a \in \Lambda$, $f \in \Sigma$. Given a formula φ we will write $Var(\varphi)$ to denote the set of variables which occur in φ.

The notion of *purely spatial* formula in $\mathsf{SL}_{\mathsf{PC}}$ is defined, as in L_{PC}, as a formula where the modal operator $\diamond a\, \_$ does not occur (but observe that here a purely spatial formula can contain variables).

To understand the choice of the sts logic $\mathsf{SL}_{\mathsf{PC}}$, note that an instance $C[p_1, \dots, p_n]$ of a given coordinator $C[X_1, \dots, X_n]$, in order to perform a transition, must match the left-hand side of the conclusion of a proof rule. This might impose the components p_i's to have a certain structure, hence the need of inserting the spatial operators $f \in \Sigma$ in the logic. Furthermore, the premises of the matched rule must be satisfiable. Such premises usually require each component p_i to be able to exhibit some behaviour, i.e. to perform a certain transition. Hence the logic includes also modal operators $\diamond a\, \_$ expressing the capability of performing action a.

The only novelty with respect to L_{PC} is the replacement of the atomic formula $\top$ with formulae X, one for each variable in $\mathcal{X}$. Satisfaction is defined as for logic L_{PC} (see Definition 7), replacing the clause for $\top$ with the following, for any process $p \in \mathcal{P}$:

$$p \models X$$

i.e., as mentioned above, a formula X, consisting of a single process variable, does not impose any constraint on the process and thus it is satisfied by any process. For instance, the formula $\diamond a\, X$ is satisfied by any process which is able to perform an action a, i.e., by any process p such that $p \rightarrow_a q$ for some q.

Variables in $\mathsf{SL}_{\mathsf{PC}}$ formulae will be later used to identify the continuation, or residual, of a process after it has exhibited the capabilities and/or structure imposed by the formula. e.g., whenever $p \models \diamond a\, X$ and thus $p \rightarrow_a q$, the variable X in the formula $\diamond a\, X$, identifies the continuation q. In the following, given a formula φ and n components $q_1, \dots, q_n$, we denote by $\varphi[q_1/X_1, \dots, q_n/X_n]$, or $\varphi[\vec{q}/\vec{X}]$ for short, the formula obtained from φ by replacing variables X_i with components q_i.

Definition 9 (Satisfaction with residuals) *Let $p \in \mathcal{P}$ be a component, let $\vec{q} = q_1, \dots, q_n$ be a tuple of components and let $\varphi \in \mathsf{SL}_{\mathsf{PC}}$ be a formula such that $Var(\varphi) \subseteq \{X_1, \dots, X_n\}$. We say that p* satisfies φ with residuals $q_1, \dots, q_n$*, and write $p \models \varphi; \vec{q}$, when*

$$p \models \varphi[q_1/X_1, \dots, q_n/X_n]$$

For example, consider the ACCS process $p \equiv n[\bar{a}.\,0 \mid a.\,b.\,0]$ and the formula $\varphi = n[\diamond\bar{a}\,X_1 \mid a.X_2]$. Then, it is easy to see that $p \models \varphi$. Moreover if $q_1 = 0$ and $q_2 = b.\,0$ then $p \models \varphi; (q_1, q_2)$.

More generally, in the following, we will use the notation $\vec{p} \models \vec{\varphi}; \vec{q}$ where $\vec{\varphi} = \varphi_1, \ldots, \varphi_k$ and $\vec{p} = p_1, \ldots, p_k$ are tuples of formulae and components, respectively, with the obvious meaning that $p_i \models \varphi_i[q_1/X_1, \ldots, q_n/X_n]$ for $i \in \{1, \ldots, k\}$.

Note that the action prefix operator yields the spatial formula $a.X$, which is satisfied by components of the shape $p \equiv a.q$. Although for specific calculi the formulae $a.X$ and $\diamond a\,X$ are satisfied exactly by the same set of components (e.g. in a trivial calculus with only the prefix operator $a.\_$ and the inactive process 0, and with just one rule $a.\ X \rightarrow_a X$), we remark that their meaning is quite different: the former imposes a structural constraint, the latter imposes a behavioural constraint, satisfied by components which can perform the action a (e.g., by the process $(b.0 \mid a.0)\backslash b$ in a CCS-like calculus).

3.2 Symbolic transition systems

We next introduce an operational description of coordinators in terms of transition systems. As mentioned before, transition labels include also a formula from the logic $\mathsf{SL}_{\mathsf{PC}}$, which imposes constraints on the unspecified components in order to allow the transition to happen. This fact will be formalised in terms of two properties (called soundness and completeness) which establish a tight connection between the symbolic transition system over coordinators and the original labeled transition system over components.

Definition 10 (Symbolic Transition System) *A* symbolic transition system *(*STS*) $\mathcal{S}$ for the process calculus* PC *is a set of transitions*

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

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 $\mathsf{SL}_{\mathsf{PC}}$ such that $\bigcup_i Var(\varphi_i) \supseteq \{Y_1, \ldots, Y_m\}$. The transition will be often written as $C[\vec{X}] \xrightarrow{\vec{\varphi}}_a D[\vec{Y}]$ and the tuple $\vec{\varphi}$ will be called the trigger.

The variable names in the states of $\mathcal{S}$ are not relevant: they are just indexed placeholders, whose number can vary along the computation. What is relevant is the correspondence between each variable X_i in the source and its residuals $Y_{i_1}, \ldots, Y_{i_h}$ in the target, as expressed by the formula φ_i, in which the residuals may occur. This could be made more formal by fixing an enumeration $X_0, X_1, \ldots$ of variables and taking the i-th variable of a coordinator to be always X_i. For the sake of readability we preferred to use different names for the variables appearing in the source and in the target of symbolic transitions.

Consider again the calculus ACCS in Example 1. A symbolic transition can be

$$n[X \mid \bar{a}.\,b.\,0] \xrightarrow{\diamond a\,Y}_\tau n[Y \mid b.\,0]$$

stating that a coordinator of the shape $n[X \mid \bar{a}.\,b.\,0]$, where X can perform an a-labelled transition becoming Y, can evolve to $n[Y \mid b.\,0]$, performing a silent step. A different symbolic transition could be

$$n[X_1] \mid X_2 \xrightarrow{Y_1, open\,n.\,Y_2}_\tau Y_1 \mid Y_2$$

$$P ::= 0 \mid \ell.\, P \mid (a)\, P$$

$$\frac{}{\ell.\, P \rightarrow_{\ell} P}\ (\mathit{pref})$$

$$\frac{P \rightarrow_{a} Q}{(a)\, P \rightarrow_{\tau} (a)\, Q}\ (\mathit{hide}) \qquad\qquad \frac{P \rightarrow_{\ell} Q}{(a)\, P \rightarrow_{\ell} (a)\, Q}\ (\mathit{lift}) \quad \ell \neq a$$

Figure 3: Syntax and operational semantics of Tick.

stating that a coordinator $n[X_1] \mid X_2$, where X_2 has the shape $\mathit{open}\, n.\, Y_2$ (i.e., it can open environment n) can evolve to $Y_1 \mid Y_2$ via a silent step (with Y_1 the same as X_1).

For $\mathcal{S}$ to provide an abstract view of PC we must of course require some additional properties enforcing the correspondence with the LTS $\mathcal{L}$ over components. Consider a transition system where coordinators have just one hole. Intuitively, whenever $C[X] \xrightarrow{\varphi}_a D[Y]$ the idea is that the coordinator C, when instantiated with any component satisfying φ, can perform action a becoming an instance of D. As explained before, the process variable Y, which typically occurs 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 $p \models \varphi; q$, 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 *soundness* and *completeness*, respectively.

Definition 11 (Soundness) *A symbolic transition*

$$C[\vec{X}] \xrightarrow{\vec{\varphi}}_a D[\vec{Y}]$$

for the process calculus PC *is called* sound *if for all tuples of components $\vec{p}$ and $\vec{q}$ such that $\vec{p} \models \vec{\varphi}; \vec{q}$, there exists a transition $C[\vec{p}] \rightarrow_a D[\vec{q}]$ in $\mathcal{L}$. An* STS *$\mathcal{S}$ for the process calculus* PC *is* sound *if all its symbolic transitions are sound.*

For instance, the aforeseen symbolic transitions for ACCS, i.e., $n[X \mid \bar{a}.\, b\, 0] \xrightarrow{\diamond a\, Y}_\tau n[Y \mid b.\, 0]$ and $n[X_1] \mid X_2 \xrightarrow{Y_1, \mathit{open}\, n.\, Y_2}_\tau Y_1 \mid Y_2$ can be easily shown to be sound in the sense above.

Definition 12 (Completeness) *An* STS *$\mathcal{S}$ for the process calculus* PC *is* complete *if for any coordinator $C[\vec{X}]$, for all tuples of components $\vec{p}$ and for any transition in $\mathcal{L}$*

$$C[\vec{p}] \rightarrow_a q$$

there exists a symbolic transition $C[\vec{X}] \xrightarrow{\vec{\varphi}}_a D[\vec{Y}]$ in $\mathcal{S}$ and a tuple of components $\vec{q}$ such that $\vec{p} \models \vec{\varphi}; \vec{q}$, and $q \equiv D[\vec{q}]$.

An example of sound and complete STS for ACCS will be provided in Section 5, since the complexity of such calculus prevents one to construct it with an ad-hoc approach. Here we consider a much simpler calculus.

Example 13 The Tick calculus is defined by the grammar and SOS operational rules in Figure 3, where ℓ ranges in a fixed set of labels Λ, $\tau \in \Lambda$ is a distinguished label and a ranges over $\Lambda - \{\tau\}$. Rule (*lift*) assumes $\ell \neq a$. Therefore processes consist of lists of actions which can be performed sequentially. The hiding operator $(a)\_$ allows to hide action a, which then shows up as τ at the top level.

Let $C[X]$ denote an arbitrary context in Tick (observe that in this simple calculus at most one variable appears in a context). Then the STS consisting of the following (schema of) symbolic transitions:

$$
\begin{aligned}
&(a_1)\ \dots (a_n)\, a.\, C[X] \xrightarrow{Y}_\tau (a_1)\ \dots (a_n)\, C[Y] \\
&(a_1)\ \dots (a_n)\, \ell.\, C[X] \xrightarrow{Y}_\ell (a_1)\ \dots (a_n)\, C[Y] \\
&(a_1)\ \dots (a_n)\, X \xrightarrow{\diamond a\, Y}_\tau (a_1)\ \dots (a_n)\, Y \\
&(a_1)\ \dots (a_n)\, X \xrightarrow{\diamond \ell\, Y}_\ell (a_1)\ \dots (a_n)\, Y
\end{aligned}
$$

where $n \geq 0$, $a \in \{a_1, \dots, a_n\}$ and $\ell \notin \{a_1, \dots, a_n\}$, is sound and complete for the calculus.

For example, the coordinator $(a)\,(b)\,a.\,X$ has just the transition

$$(a)\,(b)\,a.\,X \xrightarrow{Y}_\tau (a)\,(b)\,Y$$

while the coordinator $(a)\,(b)\,X$ has transitions

$$(a)\,(b)\,X \xrightarrow{\diamond a\, Y}_\tau (a)\,(b)\,Y \qquad (a)\,(b)\,X \xrightarrow{\diamond b\, Y}_\tau (a)\,(b)\,Y \qquad (a)\,(b)\,X \xrightarrow{\diamond \ell\, Y}_\ell (a)\,(b)\,Y$$

for $\ell \in \{a, b\}$. □

4 Symbolic bisimulations

Relying on the operational description of coordinators given by an STS we can define observational equivalences over coordinators in a direct way, without resorting to their closed instances. Here we concentrate on (strong and weak) bisimulation equivalences, but we could have considered different semantics, e.g., based on traces, as shown in [7]. Observational equivalences defined over coordinators using an STS will be shown to be coherent with the ones defined by resorting to the closed instances of coordinators.

4.1 Strict symbolic bisimulation

Given any STS we can straightforwardly define a bisimulation-like equivalence, by observing in a transition both the the ordinary label and the structural and behavioural requirements on the unspecified subcomponents expressed by the trigger.

Definition 14 (Strict Symbolic Bisimulation) *A symmetric relation $\div$ over the set of coordinators $\mathcal{C}$ is a* strict symbolic bisimulation *if for any two coordinators $C[\vec{X}]$ and $D[\vec{X}]$ such that $C[\vec{X}] \div D[\vec{X}]$, for any transition*

$$C[\vec{X}] \xrightarrow{\vec{\varphi}}_a C'[\vec{Y}]$$

there exists a transition $D[\vec{X}] \xrightarrow{\vec{\varphi}}_a D'[\vec{Y}]$ such that $C'[\vec{Y}] \div D'[\vec{Y}]$. The largest strict symbolic bisimulation is an equivalence relation called strict symbolic bisimilarity *and denoted by $\sim_s$.*

The notion of strict symbolic bisimilarity is clearly well-defined, i.e., the largest symbolic bisimulation exists and it is an equivalence relation by classical results, since $\sim_{\mathrm{s}}$ is just an ordinary bisimulation over the STS, taking as labels the pairs (trigger, label).

Strict bisimilarity requires a transition to be simulated by a transition with exactly the same trigger. All the theory could be generalised by defining two formulae φ and ψ *equivalent* if for any component p and n-tuple of components $\vec{q}$ we have

$$p \models \varphi; \vec{q} \quad \text{iff} \quad p \models \psi; \vec{q}.$$

Then one could allow a transition $C[\vec{X}] \xrightarrow{\vec{\varphi}}_a C'[\vec{Y}]$ to be simulated by a transition $D[\vec{X}] \xrightarrow{\vec{\psi}}_a D'[\vec{Y}]$, where the formulae φ_i and ψ_i are equivalent, rather than identical. Syntactic equality has been preferred to logical equivalence since, in general, the latter could be hard to verify or, even worse, undecidable. Nevertheless, given a specific calculus, equivalences which are known or easy to check can be exploited in symbolic bisimilarity (e.g., to standardise the triggers).

A first result about strict symbolic bisimilarity shows that it is coherent with respect to universal closure bisimilarity $\sim_{\mathrm{u}}$ (see Definition 5) in the sense that strict symbolic bisimilarity distinguishes as much as $\sim_{\mathrm{u}}$.

THEOREM 15 ($\sim_{\mathrm{s}} \Rightarrow \sim_{\mathrm{u}}$) *If $\mathcal{S}$ is a sound and complete* STS*, then for all coordinators $C[\vec{X}]$ and $D[\vec{X}]$*

$$C[\vec{X}] \sim_{\mathrm{s}} D[\vec{X}] \;\Rightarrow\; C[\vec{X}] \sim_{\mathrm{u}} D[\vec{X}]$$

Proof. Let $C[\vec{X}]$, $D[\vec{X}]$ be coordinators and suppose $C[\vec{X}] \sim_{\mathrm{s}} D[\vec{X}]$. We want to show that for any tuple of components $\vec{p}$, we have $C[\vec{p}] \sim D[\vec{p}]$. Let $\mathcal{R}_{\mathrm{strict}}$ be the relation over component defined as follows:

$$C'[\vec{p}] \mathcal{R}_{\mathrm{strict}} D[\vec{p}]'$$

for all $C'[\vec{X}]$, $D'[\vec{X}]$ such that $C'[\vec{X}] \sim_{\mathrm{s}} D'[\vec{X}]$ and for all tuple of components $\vec{p}$.

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

For any transition $C[\vec{p}] \rightarrow_a q$ in $\mathcal{L}$, by completeness of $\mathcal{S}$, a symbolic transition $C[\vec{X}] \xrightarrow{\vec{\varphi}}_a C'[\vec{Y}]$ and a tuple of components $\vec{r}$ exist such that $\vec{p} \models \vec{\varphi}; \vec{r}$ and $q \equiv C'[\vec{r}]$. Since $C[\vec{X}] \sim_{\mathrm{s}} D[\vec{X}]$ by hypothesis, we have that $D[\vec{X}] \xrightarrow{\vec{\varphi}}_a D'[\vec{Y}]$ with $C'[\vec{Y}] \sim_{\mathrm{s}} D'[\vec{Y}]$. By soundness of $\mathcal{S}$, and by the fact that $\vec{p} \models \vec{\varphi}; \vec{r}$, it holds that $D[\vec{p}] \rightarrow_a D'[\vec{r}]$. Since $C'[\vec{Y}] \sim_{\mathrm{s}} D'[\vec{Y}]$, we have that $C'[\vec{r}]\ \mathcal{R}_{\mathrm{strict}}\ D'[\vec{r}]$. The relation $\mathcal{R}_{\mathrm{strict}}$ is obviously symmetric and hence it is a bisimulation. Since bisimilarity $\sim$ is the largest bisimulation, it contains $\mathcal{R}_{\mathrm{strict}}$ and therefore $C[\vec{p}] \sim D[\vec{p}]$, concluding the proof. $\square$

For instance, referring to the calculus Tick, defined in Figure 3, it is not difficult to see that the coordinators $C[X] = (a)\,(b)\,X$ and $D[X] = (b)\,(a)\,X$ are strict bisimilar. In fact (the symmetric closure of) the relation $\{(C[X], D[X])\}$ is a strict symbolic bisimulation, since the symbolic moves for the coordinators are of the kind

$$C[X] \xrightarrow{\diamond\alpha\,Y}_{\ell} C[Y] \qquad\qquad D[X] \xrightarrow{\diamond\alpha\,Y}_{\ell} D[Y]$$

where $\ell = \alpha$ if $\alpha \notin \{a, b\}$ and $\ell = \tau$, otherwise. Similarly, assuming $a \neq b$, it is easy to verify that $(a)\,b.\,X \sim_{\mathrm{s}} b.\,(a)\,X$, while of course $(a)\,a.\,X \not\sim_{\mathrm{s}} a.\,(a)\,X$ and $(a)\,X \not\sim_{\mathrm{s}} (b)\,X$.

Strict symbolic bisimilarity distinguishes at least as much as universal bisimilarity, but the converse does not hold, in general. This issue is discussed later on.

Note that components are just a special case of coordinators (with no placeholders). Therefore, any sound and complete STS includes, by definition, all the transitions of the reference LTS and thus we trivially have the following result, showing that on components strict symbolic bisimilarity coincides with standard bisimilarity (which, in turn, is trivially the same as universal closure bisimilarity).

LEMMA 16 *If $\mathcal{S}$ is a sound and complete* STS*, then for all components p and q*

$$p \sim_{\mathrm{s}} q \quad \textit{iff} \quad p \sim q$$

4.2 Loose symbolic bisimulation

In certain cases, if the STS are not carefully designed, the intensional nature of $\sim_{\mathrm{s}}$ can lead to distinguish coordinators that are intuitively equivalent. In particular, this problem can arise in STS that are automatically generated starting from redundant specifications (e.g., when the same transitions of the LTS can be proved by applying different sets of rules). In this section we propose a partial solution to this problem, that can help in many situations.

In the presence of spatial formulae, the requirement of exact matching between the triggers labelling the transitions can be relaxed. Intuitively, the more liberal bisimilarity defined below allows for a transition to be simulated by another transition with weaker spatial constraints on the residuals.

In order to formalise the above intuition, it is convenient to define the composition of formulae.

DEFINITION 17 (FORMULAE COMPOSITION) *Given two tuples of formulae $\vec{\varphi} = (\varphi_1, \ldots, \varphi_n)$ and $\vec{\psi} = (\psi_1, \ldots, \psi_k)$, such that $\bigcup_i Var(\varphi_i) = \{X_1, \ldots, X_k\}$, we define*

$$\vec{\varphi}; \vec{\psi} = (\varphi_1[\vec{\psi}/\vec{X}], \ldots, \varphi_n[\vec{\psi}/\vec{X}]).$$

In words, composing $\vec{\varphi}$ and $\vec{\psi}$ we get a tuple of formulae obtained from $\vec{\varphi}$ by imposing on each variable X_i occurring in $\vec{\varphi}$, the constraint expressed by the formula ψ_i. For instance $(\diamond a\, X_1,\ X_1 \mid X_2); (n[Y_1],\ \diamond b\, Y_1) = (\diamond a\, n[Y_1],\ n[Y_1] \mid \diamond b\, Y_1)$.

In the following, whenever we write $\vec{\varphi}; \vec{\psi}$ we will implicitly assume that composition is well-defined. By simple syntactical manipulations, it is easy to prove that the composition operator over formulae ";" is associative, i.e., given tuples of formulae $\vec{\varphi}$, $\vec{\psi}$ and $\vec{\gamma}$, then $(\vec{\varphi}; \vec{\psi}); \vec{\gamma} = \vec{\varphi}; (\vec{\psi}; \vec{\gamma})$. Hence we will simply write $\vec{\varphi}; \vec{\psi}; \vec{\gamma}$. Also observe that the operator for composing formulae ";" generalises the one, denoted by the same symbol, used in the definition of satisfaction with residuals (Definition 9).

DEFINITION 18 (LOOSE SYMBOLIC BISIMULATION) *A symmetric relation $\div$ over the set of coordinators $\mathcal{C}$ is a* loose symbolic bisimulation *if for any pair of coordinators $C[\vec{X}]$ and $D[\vec{X}]$ such that $C[\vec{X}] \div D[\vec{X}]$, for any transition*

$$C[\vec{X}] \xrightarrow{\vec{\varphi}}_a C'[\vec{Y}]$$

a transition $D[\vec{X}] \xrightarrow{\vec{\psi}}_a D'[\vec{Z}]$ *and a tuple of spatial formulae* $\vec{\psi}'$ *exist such that* $\vec{\varphi} = \vec{\psi}; \vec{\psi}'$ *and* $C'[\vec{Y}] \dot{\div} D'[\vec{\psi}']$. *The greatest loose bisimulation is called* loose symbolic bisimilarity *and denoted* $\dot{\sim}_{\mathrm{l}}$.

Loose 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 loose bisimilarity $\dot{\sim}_{\mathrm{l}}$ is always coarser than strict bisimilarity $\sim_{\mathrm{s}}$.

Proposition 19 ($\sim_{\mathrm{s}} \Rightarrow \dot{\sim}_{\mathrm{l}}$) *For any symbolic transition system* $\mathcal{S}$

$$C[\vec{X}] \sim_{\mathrm{s}} D[\vec{X}] \;\Rightarrow\; C[\vec{X}] \dot{\sim}_{\mathrm{l}} D[\vec{X}].$$

Proof. It follows directly from the definition of the two bisimulations, since the spatial formulae in the tuple $\vec{\psi}'$, used in $\dot{\sim}_{\mathrm{l}}$ when simulating the step, can of course be identities. $\square$

Moreover, as witnessed by the following example, relation $\dot{\sim}_{\mathrm{l}}$ can be strictly coarser than $\sim_{\mathrm{s}}$.

Example 20 Let $\Sigma = \{a, f(.), g(.)\}$. 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 $f(X) \not\sim_{\mathrm{s}} g(X)$, 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 $f(X) \dot{\sim}_{\mathrm{l}} g(X)$. $\square$

Although coarser than $\sim_{\mathrm{s}}$, loose bisimilarity still results coherent, in the same sense discussed above for $\sim_{\mathrm{s}}$, with respect to universal closure bisimilarity $\sim_{\mathrm{u}}$.

Theorem 21 ($\dot{\sim}_{\mathrm{l}} \Rightarrow \sim_{\mathrm{u}}$) *If* $\mathcal{S}$ *is sound and complete w.r.t.* $\mathcal{L}$, *then*

$$C[\vec{X}] \dot{\sim}_{\mathrm{l}} D[\vec{X}] \;\Rightarrow\; C[\vec{X}] \sim_{\mathrm{u}} D[\vec{X}].$$

Proof. The proof is similar to, but slightly more involved than, that of Theorem 15. Let $C[\vec{X}]$, $D[\vec{X}]$ be coordinators and suppose $C[\vec{X}] \dot{\sim}_{\mathrm{l}} D[\vec{X}]$. We want to show that for any $\vec{p}$, we have $C[\vec{p}] \sim D[\vec{p}]$. Let $\mathcal{R}_{\mathrm{loose}}$ be the relation defined by

$$C[\vec{p}] \; \mathcal{R}_{\mathrm{loose}} \; D[\vec{p}] \overset{\mathrm{def}}{\iff} C[\vec{X}] \dot{\sim}_{\mathrm{l}} D[\vec{X}].$$

We first show that $\mathcal{R}_{\mathrm{loose}}$ is a bisimulation for $\mathcal{L}$. For any transition $C[\vec{p}] \rightarrow_a q$ in $\mathcal{L}$, by completeness of $\mathcal{S}$, a symbolic transition $C[\vec{X}] \xrightarrow{\vec{\varphi}}_a C'[\vec{Y}]$ and a tuple of components $\vec{r}$ exist, with $\vec{p} \models \vec{\varphi}; \vec{r}$ and $q \equiv C'[\vec{r}]$. Since $C[\vec{X}] \dot{\sim}_{\mathrm{l}} D[\vec{X}]$ by hypothesis, there must be a transition

$$D[\vec{X}] \xrightarrow{\vec{\psi}}_a D'[\vec{Z}]$$

and a tuple spatial formulae $\vec{\psi}'$ such that $C'[\vec{Y}] \dot{\sim}_{\mathrm{l}} D'[\vec{\psi}']$ and $\vec{\varphi} = \vec{\psi}; \vec{\psi}'$. Since $\vec{p} \models \vec{\varphi}; \vec{r}$, letting $\vec{s} \equiv \vec{\psi}'; \vec{r}$ it follows that $\vec{p} \models \vec{\psi}; \vec{s}$. Therefore, by soundness of $\mathcal{S}$, it follows that $D[\vec{p}] \rightarrow_a D'[\vec{s}]$. Moreover, since $C'[\vec{Y}] \dot{\sim}_{\mathrm{l}} D'[\vec{\psi}']$, we have that $C'[\vec{r}] \; \mathcal{R}_{\mathrm{loose}} \; D'[\vec{s}]$. The relation $\mathcal{R}_{\mathrm{loose}}$ is clearly symmetric and hence it is a bisimulation for $\mathcal{L}$. Since bisimilarity $\sim$ is the largest bisimulation, it contains $\mathcal{R}_{\text{loose}}$ and thus $C[\vec{p}] \sim D[\vec{p}]$. $\square$

We note that $\dot{\sim}_{\text{l}}$ is *not* guaranteed to be an equivalence relation, since it may fail to be transitive in some "pathological" situations (see the example in [7]). In such cases, its transitive closure $(\dot{\sim}_{\text{l}})^*$ should be considered.

Summarising we can conclude that both $\sim_{\text{s}}$ and $\dot{\sim}_{\text{l}}$ are meaningful. In fact $\sim_{\text{s}}$ is always an equivalence and, in view of an automated verification, the simpler formulation of $\sim_{\text{s}}$ makes it easier to check. Furthermore, since $\sim_{\text{s}}$ is the straightforward notion of bisimilarity over the STS, existing tools and techniques could be easily reusable. On the other hand $\dot{\sim}_{\text{l}}$ provides a coarser equivalence, able to deal with certain redundant symbolic transitions in a satisfactory way. Fixing the underlying calculus and the application context may help in choosing the most suitable notion.

Finally, symbolic bisimilarities can also be exploited just as convenient approximations of $\sim_{\text{u}}$. Furthermore, in using $\dot{\sim}_{\text{l}}$ as a proof technique for $\sim_{\text{u}}$, the transitivity property expressed by the following corollary of Theorem 21 can be exploited.

COROLLARY 22 *If $\mathcal{S}$ is sound and complete, then* $(\dot{\sim}_{\text{l}})^* \Rightarrow \sim_{\text{u}}$.

We conclude by pointing out that we should not expect, in general, a symbolic bisimilarity to coincide with its universal counterpart, and even for very simple calculi it may happen that $\sim_{\text{s}} \neq \sim_{\text{u}}$. Below we provide a counterexample, reworked from [38], and we try to clarify the conceptual reasons which underly this phenomenon.

EXAMPLE 23 Let us extend finite CCS with the operators $one_a(\_)$, $stop(\_)$, and with the SOS rule

$$\frac{P \to_a Q}{one_a(P) \to_a stop(Q)}$$

Although this calculus conforms to a very basic rule structure (namely, the De Simone format, see Section 5), it is not difficult to find a sound and complete STS and two universal bisimilar coordinators which are not strict symbolic bisimilar.

For instance, consider the coordinators $C[X] = a.0 + a.b.0 + a.one_b(X)$ and $D[X] = a.0 + a.b.0 + a.stop(X)$. Then it is easy to see that $C[X]$ and $D[X]$ are universal bisimilar. Indeed $C[p]$ behaves as $a.0 + a.b.0 + a.b.0$, if $p \models \diamond b Y$, or as $a.0 + a.b.0 + a.0$, otherwise. In both cases it is bisimilar to $D[p]$ that behaves as $a.0 + a.b.0 + a.0$. The following figure shows the symbolic transitions of the two coordinators in the (sound and complete) STS defined using the methodology in Section 5. Then observe that $C[X]$ and $D[X]$ are not equated by strict symbolic bisimilarity (because the transition $one_b(Y) \xrightarrow{\diamond b Z}_b stop(Z)$ cannot be simulated by any state reached from $D[X]$). Note that also $\dot{\sim}_{\text{l}} \neq \sim_{\text{u}}$ since in this case the special format of the proof rules ensures that triggers are only modal formulae or variables and thus $\dot{\sim}_{\text{l}} = \sim_{\text{s}}$.

$$
\begin{tikzcd}
 & C[X] \arrow[dl, "a"', "Y"] \arrow[d, "a"', "Y"] \arrow[dr, "Y", "a"'] & \\
0 & b.0 \arrow[d, "b"', "Z"] & one_b(Y) \arrow[d, "b"', "\diamond b\, Z"] \\
 & 0 & stop(Z)
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
 & D[X] \arrow[dl, "a"', "Y"] \arrow[d, "a"', "Y"] \arrow[dr, "Y", "a"'] & \\
0 & b.0 \arrow[d, "b"', "Z"] & stop(Z) \\
 & 0 &
\end{tikzcd}
$$

□

Intuitively this happens because instantiation is dynamic in the symbolic bisimulation game, while it is decided once and forever for $\sim_{\mathrm{u}}$. We informally claim that this situation is to some extent independent of the STS at hand and would replicate in any reasonably irredundant STS for the calculus. As already mentioned at the beginning, rather than considering this fact as a limitation for the symbolic approach, we think it clarifies the conceptual differences between our approach and that based on universal closure of coordinators.

From a more technical point of view, when sketching the proof of the possible implication $\sim_{\mathrm{u}} \Rightarrow \sim_{\mathrm{s}}$, one soon realises that $\sim_{\mathrm{u}}$ can hardly be formulated as a strict bisimilarity. Assume $C[X] \sim_{\mathrm{u}} D[X]$ (we here restrict to single-holed coordinators), and take a generic symbolic move

$$C[X] \xrightarrow{\varphi}_a C'[Y]$$

of a sound and complete STS. Then, by soundness, we know that $\forall p_i, q_i$ such that $p_i \models \varphi; q_i$ we have $C[p_i] \to_a C'[q_i]$. Then, since $C[X] \sim_{\mathrm{u}} D[X]$, for any such move, we must have $D[p_i] \to_a d_i$, with $d_i \sim C'[q_i]$. By completeness, it must be the case that there exist $\varphi_i, D'_i[Z], q'_i$ with $D[X] \xrightarrow{\varphi_i}_a D'_i[Z]$ such that $p_i \models \varphi_i; q'_i$ and $D'_i[q'_i] = d_i$, meaning that in general, according to $\sim_{\mathrm{u}}$, a symbolic move of $C[X]$ can be simulated via the joint effort of several symbolic moves of $D[X]$. More precisely, the choice of the symbolic move $D[X] \xrightarrow{\varphi_i}_a D'_i[Z]$ is dependent on the components p_i and q_i that $C[X]$ is going to use. Intuitively, the difference between the symbolic and the universal approach resembles the difference between "early" and "late" semantics, based on the time in which p_i and q_i are fixed (before the choice of transition $D[X] \xrightarrow{\varphi_i}_a D'_i[Z]$ in $\sim_{\mathrm{u}}$, after in $\sim_{\mathrm{s}}$).

The distinction between early and late is inessential provided that either (1) each formula uniquely characterises exactly one p_i and one q_i, or (2) the set of processes satisfying any two different formulae are disjoint and all symbolic transitions with the same source have different labels. Only having the calculus at hand, these semantic assumptions can be verified and eventually exploited. Finding a general way to face this issue is a challenging open problem.

4.3 Weak symbolic bisimulation

Given a calculus with a distinguished silent action τ, "weak forms" of bisimilarities can be naturally defined in our symbolic framework. Let us first define the relations $\xRightarrow{\varphi}_a$ and $\xRightarrow{\varphi}$ that represent in a single transition, called weak (symbolic) transition, a sequence of steps including or not a visible action. Formula φ, labelling the weak transitions, arises as the composition of the triggers labelling each single step.

Definition 24 (Weak Transition) *Let $\mathcal{S}$ be an* STS *for the process calculus* PC*. The weak symbolic transitions are defined as follows. For coordinators $C[\vec{X}]$ and $D[\vec{X}]$ we write*

- $C[\vec{X}] \stackrel{\vec{\varphi}}{\Longrightarrow} D[\vec{Y}]$ *if*

$$C[\vec{X}] \xrightarrow{\vec{\varphi}_1}_\tau \xrightarrow{\vec{\varphi}_2}_\tau \cdots \xrightarrow{\vec{\varphi}_h}_\tau D[\vec{Y}]$$

where $\vec{\varphi} = \vec{\varphi}_1; \ldots; \vec{\varphi}_h$, with $h \geq 0$. When $h = 0$ it is intended that $\vec{\varphi} = \vec{X}$.

- $C[\vec{X}] \stackrel{\vec{\varphi}}{\Longrightarrow}_a D[\vec{Y}]$ *if*

$$C[\vec{X}] \xrightarrow{\vec{\varphi}_1}_\tau \cdots \xrightarrow{\vec{\varphi}_{k-1}}_\tau \xrightarrow{\vec{\varphi}_k}_a \xrightarrow{\vec{\varphi}_{k+1}}_\tau \cdots \xrightarrow{\vec{\varphi}_h}_\tau D[\vec{Y}]$$

where $\vec{\varphi} = \vec{\varphi}_1; \ldots; \vec{\varphi}_h$, with $h \geq k \geq 1$.

We can now easily define weak symbolic bisimulation.

Definition 25 ($\approx_{\mathrm{w}}$) *A symmetric relation $\div$ on coordinators is a* weak symbolic bisimulation *if for all coordinators $C[\vec{X}]$, $D[\vec{X}]$ with $C[\vec{X}] \div D[\vec{X}]$*

1. *if $C[\vec{X}] \xrightarrow{\vec{\varphi}}_\tau C'[\vec{Y}]$ then $D[\vec{X}] \stackrel{\vec{\varphi}}{\Longrightarrow} D'[\vec{Y}]$ and $C'[\vec{Y}] \div D'[\vec{Y}]$;*
2. *if $C[\vec{X}] \xrightarrow{\vec{\varphi}}_a C'[\vec{Y}]$ then $D[\vec{X}] \stackrel{\vec{\varphi}}{\Longrightarrow}_a D'[\vec{Y}]$ and $C'[\vec{Y}] \div D'[\vec{Y}]$.*

The largest strict symbolic bisimulation $\approx_{\mathrm{w}}$ is an equivalence relation called weak symbolic bisimilarity

As it happens for weak symbolic bisimulation defined over components, it is immediate to see that weak symbolic bisimilarity is a well-defined relation on coordinators and it is an equivalence. Moreover, the standard relation between weak and strong bisimilarity holds.

Theorem 26 *Weak symbolic bisimilarity is coarser than strict symbolic bisimilarity, i.e., $\sim_{\mathrm{s}} \Rightarrow \approx_{\mathrm{w}}$.*

Proof. Immediate by definition. □

For instance, referring to the calculus Tick, defined in Figure 3, it is not difficult to see that the coordinators $C[X] = (a)\, a.\, X$ and $D[X] = (a)\, X$ are not strict bisimilar. Instead, they are weak bisimilar since (the symmetric closure of) the relation $\{(C[X], D[X]), (D[X], D[X])\}$ is a weak symbolic bisimulation. Roughly, this happens because the symbolic move $C[X] \xrightarrow{Y}_\tau D[Y]$ can be simulated by $D[X]$ by remaining idle.

Given a sound and complete STS $\mathcal{S}$ for a given LTS $\mathcal{L}$ we can show that soundness and completeness also extend to the relations $\stackrel{\varphi}{\Longrightarrow}$ and $\stackrel{\varphi}{\Longrightarrow}_a$. We first need a simple, but essential result on the composition of formulae.

Lemma 27 *Let $\vec{\varphi}$ and $\vec{\psi}$ be tuples of formulae and let $\vec{p}$, $\vec{r}$ be tuples of components. Then there exists a tuple of components $\vec{q}$ such that $\vec{p} \models \vec{\varphi}; \vec{q}$ and $\vec{q} \models \vec{\psi}; \vec{r}$ if and only if $\vec{p} \models \vec{\varphi}; \vec{\psi}; \vec{r}$.*

Proof. By induction on the structure of φ (see [6] for details). $\square$

PROPOSITION 28 *Let $\mathcal{S}$ be a* STS.

1. *If $\mathcal{S}$ is sound then*

   - *if $C[\vec{X}] \stackrel{\vec{\varphi}}{\Longrightarrow} D[\vec{Y}]$ then for all tuples of components $\vec{p}, \vec{q}$ such that $\vec{p} \models \vec{\varphi}; \vec{q}$, we have $C[\vec{p}] \Rightarrow D[\vec{q}]$.*
   - *if $C[\vec{X}] \stackrel{\vec{\varphi}}{\Longrightarrow}_a D[\vec{Y}]$ then for all tuples of components $\vec{p}, \vec{q}$ such that $\vec{p} \models \vec{\varphi}; \vec{q}$, we have $C[\vec{p}] \Rightarrow_a D[\vec{q}]$.*

2. *If $\mathcal{S}$ is complete then*

   - *if $C[\vec{p}] \Rightarrow q$ then there exist a symbolic transition $C[\vec{X}] \stackrel{\vec{\varphi}}{\Longrightarrow} D[\vec{Y}]$ and a tuple of components $\vec{r}$ such that $\vec{p} \models \varphi; \vec{r}$ and $D[\vec{r}] = q$.*
   - *if $C[\vec{p}] \Rightarrow_a q$ then there exist a symbolic transition $C[\vec{X}] \stackrel{\vec{\varphi}}{\Longrightarrow}_a D[\vec{Y}]$ and a tuple of components $\vec{r}$ such that $\vec{p} \models \varphi; \vec{r}$ and $D[\vec{r}] = q$.*

Proof. The proofs proceed by induction on the number of steps in a weak transition $\Longrightarrow$, exploting the compositionality result on formulae given by Lemma 27 (see [6] for details). $\square$

Then we can prove, that, as it happens for strong bisimulation, weak symbolic bisimulation implies universal weak bisimulation.

THEOREM 29 $\approx_{\mathrm{w}} \Rightarrow \approx_{\mathrm{u}}$

Proof. The proof follows exactly the same outline as that of Theorem 15, exploiting the results in Proposition 28 in place of soundness and completeness. $\square$

Although not explicitly developed here, we mention that a notion of "loose weak symbolic bisimilarity", refining weak symbolic bisimilarity, can be defined following the ideas in Section 4.2. Results analogous to those in such section are obtained along the same lines.

5 Building symbolic transitions by unification

In this section we present a constructive methodology for deriving sound and complete STSs. As explained, the trigger labelling STS transitions is intended to represent, in the *most general way*, the class of processes that enable the transition to fire, once that the coordinator has been instantiated with them, i.e., it characterises the "minimal" structure or behavioural capability needed at each step in order to make a proof rule applicable. This is clearly reminiscent of *unification* that, together with the natural interpretation of SOS proof rules as Prolog clauses, lead to the definition of a Prolog program that defines an STS for a given process calculus. Such generated STSs can be proved sound and complete.

We start by presenting the construction for calculi whose proof rules are in *algebraic* format (a quite general format including, e.g., *De Simone* format, as a special case) without structural axioms. Fixing the admissible format of the rules is needed in order to be able to derive the Prolog program.

$$\frac{\{X_i \rightarrow_{a_i} Y_i\}_{i\in I}}{f(X_1,...,X_n) \rightarrow_a D[Z_1,...,Z_n]}$$

(a) De Simone.

$$\frac{\{X_i \rightarrow_{a_i} Y_i\}_{i\in I}}{C[X_1,...,X_n] \rightarrow_a D[Z_1,...,Z_n]}$$

(b) Algebraic.

$$\frac{\{X_i \rightarrow_{a_{i,j}} Y_{i,j} \mid 1 \leq j \leq m_i\}_{i\in I}}{f(X_1,...,X_n) \rightarrow_a D[Z_1,...,Z_m]}$$

(c) GSOS.

Figure 4: SOS formats.

Characterising the desired minimality of the derived trigger is not straightforward, essentially because such minimality often results to be a local property of the chosen derivation (this is similar to what happens in the cited approaches for deriving bisimulation congruences). This problem is faced in Section 5.2 which provides a formal result of minimality for the presented construction.

In Section 6 we will illustrate how the construction can be extended to deal with structural axioms and rules in positive GSOS format.

5.1 Algebraic process calculi without structural axioms

The *algebraic* format (ALG) [25] allows a generic coordinator $C[X_1, \ldots, X_n]$, possibly involving more than one operator, to appear as left-hand side of the conclusion of the proof rules, as illustrated Figure 4(b). The premises of the rule impose behavioural constraints on some components X_i, for $i \in I \subseteq \{1, \ldots, n\}$. Then $Z_i = Y_i$ if $i \in I$ and $Z_i = X_i$ otherwise. Also $X_i \neq Y_j$ for all i, j. An *algebraic process calculus* is a process calculus whose proof rules are in algebraic format.

Observe that De Simone format [22] (see Figure 4(a)) is a just a special case of the algebraic format, where only coordinators of the kind $f(X_1, \ldots, X_n)$, involving a single operator f, are allowed to appear as left-hand side of the conclusion of a rule.

In the following, we assume that an algebraic process calculus PC without structural axioms is fixed (the refinements needed in the presence of structural axioms are discussed in Section 6.1). Recall that coordinators are required to be linear in their variables. In order to have a grasp of the problems arising when relaxing the linearity assumption, consider the ACCS coordinator $X \mid X$. Within the theory so far presented, it would be difficult to represent with a symbolic transition the transition $a.0 + \bar{a}.0 \mid a.0 + \bar{a}.0 \rightarrow_\tau 0 \mid 0$, obtained by instantiating X with $a.0 + \bar{a}.0$. In order to do that, it should be possible to specify (and derive) a sort of composite constraint for X, like $\diamond a\ Y_1 \wedge \diamond \bar{a}\ Y_2$. This will be formally dealt with, and the linearity requirement released, when treating the GSOS format (Section 6.3).

An STS for PC can be specified as a Prolog program which computes the symbolic transitions of any given coordinator.

Definition 30 (Prolog program) *The Prolog program Prog*(PC) *associated to the algebraic process calculus* PC *contains as a first clause*

```
trs(box(A,X),A,X) :- !.
```
(1)

where `box` *is a new operator, not in* Σ *and for any proof rule in* PC *the program includes a clause*

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

where $\{i_1, ..., i_k\}$ *is the set of indexes* I *of the corresponding rule and where* `Zi` $=$ `Yi`*, when* $i \in I$*, while* `Zi` $=$ `Xi` *otherwise.*

The program *Prog*(PC) defines the predicate `trs(X,A,Y)` whose intended meaning is "any component satisfying X can perform a transition labelled by A with target Y". More precisely, 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 represents a symbolic transition for the coordinator $C[X_1, \ldots, X_n]$ with action label a: the computed answer substitutions for the variables `X1`, ..., `Xn` represent the formulae in $\mathsf{SL}_{\mathsf{PC}}$ labelling the transition and `Z` the target coordinator. (We assume identity substitution for those variables in $X_1, \ldots, X_n$ not appearing in the coordinator, like X_2 in $C[X_1, X_2] = X_1$).

The first clause in *Prog*(PC) is used only in the refutation of goals of the kind `trs(X,a,_)` whose first argument is a variable (since `box` is not an operator in PC). In this case there is no need to impose structural requirements on X, in fact the only requirement for any component `X` for doing `a` and becoming `Y` is exactly $\diamond a\, Y$, represented in the program as `box(a,Y)`. Thus the goal can be proved by just imposing such behavioural constraint on the component corresponding to `X`. The cut operator in the body of clause (1) avoids that subsequent refutations are tried, using different clauses that could be otherwise matched by the goal `trs(X,a,_)`. To this aim, it is important that clause (1) is listed in *Prog*(PC) before all the other clauses.

The second class of clauses in *Prog*(PC) just represent a Prolog translation of the operational proof rules of the calculus. Each such clause imposes (by unification) the most general structural (spatial) constraints that the unspecified components of a coordinator should satisfy in order to make the proof rule applicable. 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 STS for the process calculus PC (over logic $\mathsf{SL}_{\mathsf{PC}}$) specified by the Prolog program *Prog*(PC) is sound and complete for the considered calculus.

THEOREM 31 *The* STS *specified by Prog*(PC) *is sound and complete.*

Proof. The proof is divided in two separate parts. The first part (soundness) is worked out in full detail, while for the sake of readability, the second part (completeness) is presented at an higher level of description.

Soundness. To prove *soundness* observe that a symbolic transitions

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

belongs to the STS if a refutation of the query

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

with computed answer substitution `Xi = ` φ_i and `Z = D[Y1, ..., Yh]` exists. By induction on length ℓ of the refutation, we prove that for any $\vec{q}$ and $\vec{p}$ such that $\vec{p} \models \vec{\varphi}; \vec{q}$ a derivation for the transition $C[p_1, \ldots, p_n] \rightarrow_a D[q_1, \ldots, q_h]$ exists.

- ($\ell = 1$) We distinguish two possibilities:
  - The query is unified with the Prolog clause `trs(box(A,Y),A,Y) :- !.` and thus $C[\vec{X}]$ is a variable, say X.
    In this case the computed answer substitution contains `X = box(a, Y)`, which represents a symbolic transition $X \xrightarrow{\diamond a Y}_a Y$. By definition of $\diamond$, for any process q such that $p \models \diamond a q$, it holds
    $$p \rightarrow_a q.$$
  - The query unifies with a clause having empty body
    ```
    trs(E[W1,...,Wm], a, F[W1,...,Wm]).
    ```
    arising from a proof rule with empty premises
    $$\frac{}{E[\vec{W}] \rightarrow_a F[\vec{W}]} \tag{2}$$
    Note that since none of the W_i is tested in the premises, the variables occurring in the target coordinator are again the W_i's.
    The most general unifier will be split as $\vec{\varphi}$ over $X_1, \ldots, X_n$ and $\vec{\psi}$ over $W_1, \ldots, W_m$, both consisting only of purely spatial formulae:
    $$C[\vec{\varphi}] = E[\vec{\psi}]$$
    and, moreover, $D[\vec{Y}] = F[\vec{\psi}]$.
    Now, for any $\vec{p}, \vec{q}$ such that $\vec{p} \models \vec{\varphi}; \vec{q}$, since φ is purely spatial, we have $\vec{p} = \vec{\varphi}; \vec{q}$ and thus
    $$C[\vec{p}] = C[\vec{\varphi}; \vec{q}] = C[\vec{\varphi}]; \vec{q} = E[\vec{\psi}]; \vec{q}$$
    Therefore, by using the proof rule (2), we obtain the desired transition
    $$C[\vec{p}] = E[\vec{\psi}]; \vec{q} \rightarrow_a F[\vec{\psi}]; \vec{q} = D[\vec{Y}]; \vec{q} = D[\vec{q}].$$
- ($\ell > 1$) In this case the successful refutation that generates the symbolic transition for $C[\vec{X}]$ starts by unifying the query with the head of a clause, say

  ```
  trs(E[W1,...,Wm], a, F[Z1,...,Zm]) :-  trs(Wi1, ai1, Ui1),...,
                                         trs(Wik, aik, Uik).
  ```

  where we recall that $\{i_1, ..., i_k\}$ is the set of indexes I of the corresponding rule and where `Zi` is either `Ui`, when $i \in I$, or `Wi` otherwise. The unification gives a most general unifier $\vec{\psi}$ over `X1,...,Xn,W1,...,Wm`, which consists only of purely spatial formulae (that can be read as coordinators).

  By hypothesis, we have successful refutations for any `trs(Wj;`$\vec{\psi}$`, aj, Uj)` for $j \in I$, with computed answer substitutions $\vec{\xi}$. We call $\vec{\theta}_j$ the restriction of $\vec{\xi}$

to the variables in $\mathtt{Wj};\vec{\psi}$ and η_j the restriction to `Uj` (we split the computed answer substitution just for convenience of notation in the rest of the proof). Note that due to the rule format η_j is purely spatial and that by linearity the variables appearing in each subgoal are all disjoint. Hence the refutations for the subgoals are "independent", in the sense that they produce substitutions for distinct variables of the original goal.

Then, in the transition

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

we have $\varphi_i = X_i; \vec{\psi}; \vec{\xi}$ and $D[\vec{Y}] = F[\vec{Z}]; \vec{\psi}; \vec{\xi} = F[\vec{Z}; \vec{\psi}; \vec{\xi}]$ (since the formulae $Z_i; \vec{\psi}; \vec{\xi}$ are purely spatial).

Each successful refutation of $\mathtt{trs(Wj;}\vec{\psi}\mathtt{,\ aj,\ Uj)}$ determines the symbolic transition

$$W_j; \vec{\psi} \xrightarrow{\vec{\theta}_j}_{a_j} U_j; \vec{\xi} = U_j; \eta_j$$

Since the length of such refutations is clearly less than ℓ, by inductive hypothesis, $\forall \vec{u}_j, \vec{v}_j$ such that $\vec{u}_j \models \vec{\theta}_j; \vec{v}_j$ the ground transition

$$W_j; \vec{\psi}; \vec{u}_j \rightarrow_{a_j} U_j; \eta_j; \vec{v}_j$$

exists. These transitions satisfy the premise of the proof rule which corresponds to the selected Prolog rule. Moreover, it also holds

$$E[r_1, \ldots, r_m] \rightarrow_a F[s_1, \ldots, s_m]$$

whenever $r_j \models W_j; \vec{\psi}; \vec{u}_j$ and $s_j \models U_j; \eta_j; \vec{v}_j = U_j; \vec{\xi}; \vec{v}_j$ if $j \in I$, while $r_i = s_i$ is any process otherwise.

Now, we have to show that for any $\vec{p}, \vec{q}$ such that $\vec{p} \models \vec{\varphi}; \vec{q}$ we can find suitable $\vec{r}, \vec{s}$ such that

$$C[p_1, \ldots, p_n] = E[r_1, \ldots, r_m] \rightarrow_a F[s_1, \ldots, s_m] = D[q_1, \ldots q_h]$$

Since $\vec{\varphi} = \vec{X}; \vec{\psi}; \vec{\xi}$ we have $\vec{p} \models \vec{X}; \vec{\psi}; \vec{\xi}; \vec{q}$. The formulae $\vec{X}; \vec{\psi}$ are purely spatial, hence the residual $\vec{t}$ of $\vec{p}$ after satisfying $\vec{X}; \vec{\psi}$ is uniquely determined, and $\vec{t} \models \vec{\xi}; \vec{q}$. Then, by associativity and by definition of $\vec{\psi}$:

$$\begin{aligned} C[\vec{p}] &= C[\vec{X}; \vec{\psi}; \vec{t}] \\ &= C[\vec{X}]; \vec{\psi}; \vec{t} \\ &= E[\vec{W}]; \vec{\psi}; \vec{t} \\ &= E[\vec{W}; \vec{\psi}; \vec{t}] \end{aligned}$$

Then let $\vec{r} = \vec{W}; \vec{\psi}; \vec{t}$ and $s_j = r_j$ if $j \notin I$, while $s_j = U_j; \vec{q}$ otherwise. For $j \in I$, let also $\vec{u}_j$ be the residual of r_j after satisfying $W_j; \vec{\psi}$, i.e., $r_j = W_j; \vec{\psi}; \vec{u}_j$, and let $\vec{v}_j$ be the residual of r_j after satisfying $W_j; \vec{\psi}; \vec{\xi}$. Then $\vec{u}_j \models \vec{\theta}_j; \vec{v}_j$ and therefore we can prove that

$$C[p_1, \ldots, p_n] = E[r_1, \ldots, r_m] \rightarrow_a F[s_1, \ldots, s_m]$$

Recalling that $D[\vec{Y}] = F[\vec{Z}]; \vec{\psi}; \vec{\xi}$, the proof is completed by showing that

$$\begin{aligned} D[\vec{q}] &= D[\vec{Y}; \vec{q}] \\ &= D[\vec{Y}]; \vec{q} \\ &= F[\vec{Z}]; \vec{\psi}; \vec{\xi}; \vec{q} \\ &= F[\vec{Z}; \vec{\psi}; \vec{\xi}; \vec{q}] \\ &= F[\vec{s}] \end{aligned}$$

Completeness. Let $C[\vec{X}]$ be a coordinator and let $\vec{p}$ be a tuple of components and q a component such that $C[\vec{p}] \rightarrow_a q$. We have to show that the goal

$$\texttt{?- trs(C[}\vec{X}\texttt{], a, Z).}$$

returns a computed answer $\vec{X} = \vec{\varphi}$ and $Z = D[\vec{Y}]$ such that $\exists \vec{r}$ with $\vec{p} \models \vec{\varphi}; \vec{r}$ and $D[\vec{r}] = q$.

Observe that since $C[\vec{p}] \rightarrow_a q$ there exists a refutation in $\mathit{Prog}(\mathsf{PC})$ of the goal

$$\texttt{?- trs(C[}\vec{p}\texttt{], a, q).}$$

which does not use clause (1). Let us proceed by induction on the length ℓ of such refutation.

- ($\ell = 1$) We distinguish two possibilities:
  - If the coordinator $C[\vec{X}]$ is a variable, say X, then the query

    $$\texttt{?- trs(X, a, Z).}$$

    returns $\texttt{X} = \diamond\texttt{a Y}$ and $\texttt{Z} = \texttt{Y}$. Clearly $p \models \diamond aq$ (and $\texttt{Z}; \texttt{q} = \texttt{q}$).
  - Assume that $C[\vec{X}]$ does not consist of a single variable. Since the query `?- trs(C[`$\vec{p}$`], a, q).` is refuted in one step, it must unify with a clause with empty body

    $$\texttt{trs(E[W1,...,Wm], a, F[W1,...,Wm]).} \tag{3}$$

    arising from a proof rule with empty premises

    $$\frac{}{E[\vec{W}] \rightarrow_a F[\vec{W}]} \tag{4}$$

    Since clause (3) instantiates to `trs(C[`$\vec{p}$`], a, q).` there will be a most general unifier of $C[\vec{X}]$ and $E[\vec{W}]$ that can be split in $\vec{\psi}$, $\vec{\theta}$ for $\vec{X}$, $\vec{W}$, respectively, such that $C[\vec{\psi}] = E[\vec{\theta}]$. Moreover, $\vec{\psi}$, $\vec{\theta}$ can be seen as purely spatial formulae. We also have a tuple of components $\vec{r}$ such that

    $$\vec{\psi}; \vec{r} = \vec{p} \qquad E[\vec{\theta}; \vec{r}] = C[\vec{\psi}; \vec{r}] = C[\vec{p}] \qquad F[\vec{\theta}; r] = q.$$

    The above implies that there is a refutation of `?- trs(C[`$\vec{X}$`], a, Z).` with computed answer substitution $\vec{\psi}$ for $\vec{X}$ and $F[\vec{\theta}]$ for Z, corresponding to a symbolic transition

    $$C[\vec{X}] \xrightarrow{\vec{\psi}}_a F[\vec{\theta}]$$

which is the desired transition. In fact, $\vec{p} \models \vec{\psi}; \vec{r}$ and $q = F[\vec{\theta}; r] = F[\vec{\theta}]; r$.

- ($\ell > 1$) If $C[\vec{X}]$ is a variable, say X, we conclude as in the first case of the previous point. Otherwise take the first clause (corresponding to a proof rule of PC), used in the refutation of `trs(C[`$\vec{p}$`], a, q)`, say

$$\mathtt{trs(E[\vec{W}], a, F[\vec{Z}])} : -\mathtt{trs(\vec{W}, \vec{a}, \vec{Z})}. \tag{5}$$

where, $\overline{\mathtt{trs}}(\vec{\mathtt{W}}, \vec{\mathtt{a}}, \vec{\mathtt{Z}})$. is a shortcut for

$$\mathtt{trs(W1, a1, Z1)}, \dots, \mathtt{trs(Wn, an, Zn)}.$$

and, to simplify the notation we are assuming that all variables are tested in the SOS rule (the general case would be completely analogous).

Since the head of the clause (5) instantiates to `trs(C[`$\vec{p}$`], a, q)`, there will be a most general unifier between $C[\vec{X}]$ and $E[\vec{W}]$ that can be split in $\vec{\psi}$, $\vec{\theta}$ for $\vec{X}$, $\vec{W}$, respectively, such that $C[\vec{\psi}] = E[\vec{\theta}]$. Moreover, $\vec{\psi}$ and $\vec{\theta}$ can be seen as purely spatial formulae. We also have tuples of components $\vec{r}$ and $\vec{s}$ such that

$$\vec{\psi}; \vec{r} = \vec{p} \qquad E[\vec{\theta}; \vec{r}] = C[\vec{\psi}; \vec{r}] = C[\vec{p}] \qquad F[\vec{s}] = q.$$

The original ground goal is thus reduced to the following ground subgoals obtained by instantiating the body of clause (5)

$$\overline{\mathtt{trs}}(\vec{\theta}; \vec{\mathtt{r}}, \vec{\mathtt{a}}, \vec{\mathtt{s}}).$$

whose refutation will be of length shorter than ℓ. Hence, by inductive hypothesis, the Prolog program will compute symbolic transitions

$$\theta_i \xrightarrow{\vec{\gamma}_i}_{a_i} G_i[\vec{Y}_i]$$

such that, for any i there exists a tuple of components $\vec{u}_i$ satisfying $\vec{r}_i \models \vec{\gamma}_i; \vec{u}_i$, where $\vec{r}_i$ consists of a suitable subset of components of $\vec{r}$ and $G_i[\vec{u}_i] = s_i$. Note that the linearity assumption on coordinators ensures that the sets of variables in different coordinators $\vec{\theta}_i$ are disjoint.

Therefore, coming back to the original coordinator, the Prolog program will determine the symbolic transition

$$C[\vec{X}] \xrightarrow{\vec{\psi}; \vec{\gamma}}_a F[\vec{G}[\vec{Y}]]$$

where $\vec{\gamma} = (\vec{\gamma}_1, \dots, \vec{\gamma}_k)$ and $\vec{G}[\vec{Y}] = G_1[\vec{Y}_1], \dots, G_k[\vec{Y}_k]$.

To conclude, recall that $\vec{p} = \vec{\psi}; \vec{r}$ and $\vec{r}_i \models \vec{\gamma}_i; \vec{u}_i$, for all i. Hence $\vec{p} \models \vec{\psi}; \vec{\gamma}; \vec{u}$, where $\vec{u} = (\vec{u}_1, \dots, \vec{u}_k)$ and $F[\vec{G}[\vec{u}]] = q$, as desired. □

Observe that, according to Definition 30, any proof rule of the calculus PC gives rise to Prolog clauses in *Prog*(PC) where the action label is a ground term. While this simplifies the theoretical treatment, for practical purposes the proof rules that are parametric in action labels can be represented by Prolog clause containing variables. For instance, considering the proof rule

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

all the corresponding clauses in *Prog*(PC)

```
trs(X1|X2, ai, Y1|X2) :- trs(X1, ai, Y1).
```

with `ai` ranging in the set of labels, can be replaced by the following single clause

```
trs(X1|X2, A, Y1|X2) :- trs(X1, A, Y1).
```

In this way, the Prolog program can be finite whenever the semantics of the calculus is defined by a finite set of rule schemata. This approach will be used in the examples which follow.

When querying the program, the variable `A` can either be instantiated, like in `?- trs(X, b, Y).` (asking under which conditions `X` can perform an observable action `b`) or left uninstantiated, like in `?- trs(X, A, Y).` (asking under which conditions `X` can perform any action). In this case, `A`, as well as the variables possibly occurring in the computed answers, play the role of a "symbolic" variable ranging over the set of actions of the calculus.

The backtracking mechanism of Prolog and the use of meta-logic operators (like `bagof` and `findall`) allow one to determine all the symbolic transitions for any coordinator. The answers are finitely many and they can be computed in a finite amount of time under the assumption that the set of SOS rules of the calculus is finite.

EXAMPLE 32 This example shows how ground transitions can be abstracted by symbolic ones, and how, once that the coordinator has been fixed, the symbolic transitions are generated. This reconsiders the main aspects of the proof of Theorem 31. We consider a simple fragment of CCS with prefix and parallel composition only, i.e., the syntax of processes is

$$P \ ::= \ 0 \ \mid \ \alpha.\, P \ \mid \quad P \mid P$$

where $\alpha \in \mathbf{Act} = \{a, \bar{a} \mid a \in \mathbf{A}\}$, for a given set of actions $\mathbf{A}$. The semantics is defined by the following SOS rules:

$$\frac{}{\alpha.\, X \to_\alpha X} \qquad \frac{X_1 \to_\alpha Y_1 \quad X_2 \to_{\bar{\alpha}} Y_2}{X_1 \mid X_2 \to_\tau Y_1 \mid Y_2}$$

$$\frac{X_1 \to_\ell Y_1}{X_1 \mid X_2 \to_\ell Y_1 \mid X_2} \qquad \frac{X_2 \to_\ell Y_2}{X_1 \mid X_2 \to_\ell X_1 \mid Y_2}$$

where ℓ ranges in $\mathbf{Act} \cup \{\tau\}$.

A Prolog program for the calculus can be the following:

```
trs(box(A,Y),A,Y):- !.
trs(A.X1, A, X1).
trs(X1|X2, tau, Y1|Y2) :- trs(X1, A, Y1), trs(X2, 'A, Y2).
trs(X1|X2, A, Y1|X2) :- trs(X1, A, Y1).
trs(X1|X2, A, X1|Y2) :- trs(X2, A, Y2).
```

where `'A` stands for the complementary action of `A`.

The transition $a.\, b.\, 0 \mid \bar{a}.\, 0 \to_\tau b.\, 0 \mid 0$ can be abstracted at the symbolic level, amongst other possibilities, by reading the component $a.\, b.\ 0 \mid \bar{a}.\ 0$ as an instance of the coordinator $X_1 \mid X_2$, through the component tuple $\vec{p} = (a.\, b.\, 0,\ \bar{a}.\, 0)$. Then the query

```
?- trs(X1|X2, tau, Z).
```

unifies with the second Prolog clause, leading to the answer `Z=Y1|Y2`. Each one of the atoms in the body unifies in turn with the first clause of the program, yielding the computed answers `box(A,Y1)` and `box('A,Y2)` for `X1` and `X2` respectively. This corresponds to the symbolic transitions below, where a ranges over the set of action

$$X_1 \mid X_2 \xrightarrow{\diamond a\, Y_1, \diamond \bar{a}\, Y_2}_{\tau} Y_1 \mid Y_2$$

Note that $\vec{p} \models (\diamond a\, Y_1, \diamond \bar{a}\, Y_2); \vec{q}$, where $\vec{q} = (b.0,\ 0)$ and a is a generic action, showing how the original ground transition arises as an instance of the symbolic transition above.
On the other hand, the above component can also be understood as an instance of the coordinator $a.X \mid \bar{a}.0$ where X is replaced by the component $p = b.0$. In this case, the query

```
?- trs(a.W|'a.0, tau, Z).
```

unifies with the third Prolog clause, with `Z=Y1|Y2` as computed substitution. Then, both the atoms `trs(a.W, A, Y1).` and `trs('a.0, 'A, Y2).` unify with the second clause of the program, yielding the computed answers `Y1=W` and `Y2=0`. This defines the symbolic transition

$$a.X \mid \bar{a} \xrightarrow{Y}_{\tau} Y \mid 0$$

□

5.2 Minimality

The use of unification for the generation of the STS naturally suggests that such STS satisfies some minimality property. We next provide a result aimed at formalising such intuition.

Let us start by observing that we cannot expect that given a symbolic transition

$$C[\vec{X}] \xrightarrow{\vec{\varphi}}_{a} D[\vec{Y}] \tag{6}$$

the label $\vec{\varphi}$ expresses the minimal constraints on $\vec{X}$ which allow the coordinator to perform a generic a-labelled step. In fact, in general, according to the operational semantics of a process calculus PC, there can be several different rules which can be applied to let the coordinator $C[\vec{X}]$ evolve. For instance, consider a set of proof rules including

$$\frac{Y \to_b Z}{f(X,Y) \to_a h(X,Z)}\ (r_1) \qquad\qquad \frac{Y \to_b Z}{f(g(X),Y) \to_a h(X,Z)}\ (r_2)$$

Then the STS computed by the $\mathit{Prog}(\mathsf{PC})$ program contains, for a coordinator $f(X,Y)$, the two symbolic transitions

$$f(X,Y) \xrightarrow{X',\ \diamond b\, Z}_{a} h(X',Z) \qquad\qquad f(X,Y) \xrightarrow{g(X'),\ \diamond b\, Z}_{a} h(X',Z)$$

Note that both transitions are needed in order to have a complete STS. In fact, in a scenario in which $b.0 \to_b 0$, the first symbolic transition alone would not represent the transition $f(g(0), b.0) \to_a h(0,0)$ (since it does not "consume" $g(\_)$). The second symbolic transition, labeled with the trigger $(g(X'),\ \diamond b\, Y)$, is hence necessary despite of the presence of the first transition, with the "smaller" label (X', $\diamond b\, Y$), i.e. a label imposing a weaker constraint.

Instead, if (6) is a transition of the generated STS, the label $\vec{\varphi}$ expresses the minimal requirements on the unspecified components $\vec{X}$ which allow the transition of $C[\vec{X}]$ to happen according to a specific (fragment of) refutation, or equivalently, SOS derivation. Intuitively speaking, let d be the fixed initial fragment of a refutation involving $C[\vec{X}]$. By (the proof of) correctness of the STS generated by *Prog*(PC), we know that for all components $\vec{p}$, $\vec{q}$ such that $\vec{p} \models \vec{\varphi}; \vec{q}$ we have

$$C[\vec{p}] \to_a D[\vec{q}]$$

with a refutation (SOS derivation) which "extends" d. The minimality result below shows that the satisfaction of $\vec{\varphi}$ is not only sufficient, but also *necessary* for the components $\vec{p}$, $\vec{q}$ to make *such* a derivation possible, and thus $\vec{\varphi}$ cannot be weakened. For instance, reconsider the symbolic transition $f(X,Y) \xrightarrow{g(X'),\ \diamond b\, Z}_a h(X', Z)$. In this case the fixed initial fragment d of SOS derivation consists only of rule r_2. Then for all pair of components p, p' such that $f(p,p') \to_a s$ with a derivation which extends d, i.e., which starts by using rule r_2, we necessarily have that $p \models g(X'); q$ and $p' \models \diamond b\, Z; q'$ for suitable q and q'.

It is worth noting that this is conceptually similar to what happens in the mentioned "deriving bisimulation congruences" approach, started with [43, 33], where, roughly speaking, the labels of transitions $p \xrightarrow{C[\_]} q$ do not represent the minimal context which allows for a generic reaction, but rather the minimal context which allow a fixed reduction rule to be applied at a fixed position.

In order to formalise these concepts, let us first introduce for any refutation of a goal in *Prog*(PC) the corresponding derivation tree, which is simply a tree (represented as a term), where nodes are labelled by the clauses used. To this aim we will denote the clauses in *Prog*(PC) by $r_0, r_1, \ldots, r_n$, where r_0 is assumed to be clause (1) of Definition 30.

DEFINITION 33 (DERIVATION TREE FOR A REFUTATION) *Let d be the refutation of a goal* $?-\mathtt{trs}(\mathtt{C}[\mathtt{X1},\ldots,\mathtt{Xn}],\mathtt{A},\mathtt{Z})$ *in Prog*(PC). *The* derivation tree *of d, denoted by* $\mathsf{Tr}(d)$*, is defined as* $\mathsf{Tr}(d) = r_i(\mathsf{Tr}(d_1), \ldots, \mathsf{Tr}(d_n))$*, where r_i is the first clause used in the refutation, and $d_1, \ldots, d_n$ is the (possibly empty) list of the refutations of the subgoals spawned by the use of rule r_i.*

We next define an order on the derivation trees.

DEFINITION 34 (ORDER ON DERIVATION TREES) *The order on derivation trees is inductively defined as*

- $r_0 \preceq T$ *for any derivation tree T and*
- *if* $T_i \preceq T'_i$ *for* $i \in \{1, \ldots k\}$ *then* $r_i(T_1, \ldots, T_k) \preceq r_i(T'_1, \ldots, T'_k)$.

When $T_1 \preceq T_2$ *we say that* T_2 extends T_1.

Intuitively, the order $\preceq$ is a kind of prefix ordering. Roughly, $T_1 \preceq T_2$ when T_2 can be obtained from T_1 by replacing the application of clause r_0, which just imposes that a variable component X is able to perform an a-labelled transition, with a derivation explicitly proving how such a-transition can be performed by suitably instantiating X.

We can finally formalise the minimality result informally discussed above.

Proposition 35 (minimality) *Let* $C[\vec{X}] \xrightarrow{\vec{\varphi}}_a D[\vec{Y}]$ *be a symbolic transition generated by* $Prog(\mathsf{PC})$ *with a refutation* d*. For all components* $\vec{p}$*,* s*, if* $C[\vec{p}] \rightarrow_a s$ *is proved by a refutation* d' *satisfying* $\mathsf{Tr}(d) \preceq \mathsf{Tr}(d')$ *then there exists a tuple of components* $\vec{q}$ *such that* $\vec{p} \models \vec{\varphi}; \vec{q}$ *and* $s = D[\vec{q}]$*.*

Proof. The proof proceeds by induction on the height ℓ of $\mathsf{Tr}(d)$.

($\ell = 1$) In this case $\mathsf{Tr}(d) = r$ for some clause r in $Prog(\mathsf{PC})$. We distinguish two subcases according to the clause r used in the refutation.

- If $r = r_0$ then the coordinators $C[\vec{X}]$ and $D[Y]$ are variables, X and Y respectively say, and $\vec{\varphi} = \diamond a\, Y$. Now, for all p, q such that $C[p] = p \rightarrow_a s$, by taking $q = s$ we have $s = D[q]$ and $p \models \varphi; q$ trivially satisfied.
- If $r \neq r_0$, then r must be a clause with empty body (corresponding to a rule without premises) of the kind

```
trs(E[W1,...,Wm], a, F[W1,...,Wm]).
```

  and the most general unifier between $C[\vec{X}]$ and $E[\vec{W}]$ is $\vec{\varphi}$ on $\vec{X}$ and some other substitution $\vec{\eta}$ over $\vec{W}$ such that

$$C[\vec{\varphi}] = E[\vec{\psi}]$$

  and, additionally, $D[\vec{Y}] = F[\vec{\psi}]$.

  Now, if $C[\vec{p}] \rightarrow_a s$ with a derivation d' such that $\mathsf{Tr}(d) \preceq \mathsf{Tr}(d')$, it is easy to see that $\mathsf{Tr}(d') = \mathsf{Tr}(d) = r$, i.e., the refutation d' uses only clause r. This means that there is a unifier between $C[\vec{p}]$ and $E[\vec{W}]$, i.e., a substitution $\vec{r}$ such that $C[\vec{p}] = E[\vec{r}]$. Moreover $F[\vec{r}] = s$.

  By the properties of the most general unifier, there must be $\vec{q}$ such that

$$\vec{r} = \vec{\psi}; \vec{q}$$

  Therefore we have

$$s = F[\vec{r}] = F[\vec{\psi}; \vec{q}] = F[\vec{\psi}]; \vec{q} = D[\vec{Y}]; \vec{q} = D[\vec{q}].$$

  Moreover

$$C[\vec{p}] = E[\vec{r}] = E[\vec{\psi}; \vec{q}] = E[\vec{\psi}]; \vec{q} = C[\vec{\varphi}]; \vec{q} = C[\vec{\varphi}; \vec{q}],$$

  hence $\vec{p} = \vec{\varphi}; \vec{q}$, which by definition implies that, as desired

$$\vec{p} \models \vec{\varphi}; \vec{q}.$$

($\ell > 1$) In this case $\mathsf{Tr}(d) = r(T_1, \ldots, T_k)$ and thus the refutation which produces the symbolic transition for $C[\vec{X}]$ starts by unifying the query with the head of a clause r, say

```
trs(E[W1,...,Wk], a, F[Z1,...,Zk]) :-  trs(W1, a1, Z1),...,
                                        trs(Wk, ak, Zk).
```

where, to simplify the notation, we are assuming that all variables are tested in the SOS rule (the general case would be analogous). The unification gives a most general unifier $\vec{\psi}$, $\vec{\zeta}$ over $\vec{X}$, $\vec{W}$, respectively, which consists only of purely spatial formulae (that can be read as coordinators). Hence

$$C[\vec{\psi}] = E[\vec{\zeta}] \tag{7}$$

Moreover, we have successful refutations d_i for any `trs(Wi;`$\vec{\zeta}$`, ai, Zi)`, with computed answer substitutions $\vec{\theta_i}$ for the variables in `Wi;`$\vec{\zeta}$ and η_i for `Zi` (as usual we split the substitution for notational convenience). Each refutation d_i, whose derivation tree is $\mathsf{Tr}(d_i) = T_i$, determines a symbolic transition

$$W_i; \vec{\zeta} \xrightarrow{\vec{\theta_i}}_{a_i} = Z_i; \eta_i. \tag{8}$$

Additionally, if we denote by $\vec{\theta}$, $\vec{\eta}$ the unions of all $\vec{\theta_i}$ and η_i, respectively, in the transition

$$C[X_1, ..., X_n] \xrightarrow{\varphi_1 \dots \varphi_n}_a D[Y_1, ..., Y_h]$$

we have $\vec{\varphi} = \vec{\psi}; \vec{\theta}$ and $D[\vec{Y}] = F[\vec{Z}]; \vec{\eta} = F[\vec{\eta}]$.

Now, let $C[\vec{p}] \rightarrow_a s$, i.e., the goal `trs(C[`$\vec{p}$`], a, s)` is provable with a refutation d' such that $\mathsf{Tr}(d') \preceq \mathsf{Tr}(d) = r(T_1, \dots, T_k)$. Then $\mathsf{Tr}(d') = r(T_1', \dots, T_k')$, where $T_i \preceq T_i'$. Since the refutation d' starts by using clause r, there are substitutions (which are actually tuple of components) $\vec{r}$ and $\vec{t}$ such that

$$C[\vec{p}] = E[\vec{r}] \qquad \text{and} \qquad s = F[\vec{t}] \tag{9}$$

Since $\vec{\psi}$ is the most general unifier of $C[\vec{X}]$ and $E[\vec{W}]$, there must be $\vec{p'}$ and $\vec{r'}$ such that

$$\vec{p} = \vec{\psi}; \vec{p'} \qquad \text{and} \qquad \vec{r} = \vec{\zeta}; \vec{r'} \tag{10}$$

The subgoal after applying clause r are in this case:

`trs(Wi;`$\vec{r}$`, ai, Zi;`$\vec{t}$`)` $=$ `trs(Wi;`$\vec{\zeta}; \vec{r'_i}$`, ai, Zi;`$\vec{t}$`)`

where $\vec{r'_i}$ denotes the components of $\vec{r'}$ over the variables in `Wi;`$\vec{\zeta}$. These goals have successful refutations d_i', such that $\mathsf{Tr}(d_i') = T_i'$, which correspond to transitions $W_i; \vec{\zeta}; \vec{r'_i} \rightarrow_{a_i} Z_i; \vec{t}$. Recalling that the derivation tree associated to the refutation producing the symbolic transition (8) is $T_i \preceq T_i'$, we can apply the inductive hypothesis and deduce that for all i there exists $\vec{q_i}$ such that

$$Z_i; \eta_i; \vec{q_i} = Z_i; \vec{t} \qquad \text{and} \qquad \vec{r'_i} \models \theta_i; \vec{q_i} \tag{11}$$

If we denote by $\vec{q}$ the union of the $\vec{q_i}$'s we have that

$$s = F[\vec{t}] = F[\vec{Z}; \vec{t}] = F[\vec{Z}; \vec{\eta}; \vec{q}] = F[\vec{Z}; \vec{\eta}]; \vec{q} = D[\vec{Y}]; \vec{q} = D[\vec{q}]$$

as desired.

Moreover, from the second part of (11) we have that

$$\vec{r'} \models \vec{\theta}; \vec{q} \tag{12}$$

Therefore we have:

$$
\begin{aligned}
C[\vec{p}] = & & \text{[by (9)]}\\
&= E[\vec{r}] & \text{[by (10)]}\\
&= E[\vec{\zeta}; \vec{r'}] & \text{[by (12) and the fact that } \models \text{, by definition,}\\
& & \text{is closed under contextualisation]}\\
&\models E[\vec{\zeta}; \vec{\theta}; \vec{q}] & \text{[by (7)]}\\
&= C[\vec{\psi}; \vec{\theta}; \vec{q}] & \text{[since } \vec{\varphi} = \vec{\psi}; \vec{\theta}\text{]}\\
&= C[\vec{\varphi}; \vec{q}]
\end{aligned}
$$

Therefore $C[\vec{p}] \models C[\vec{\varphi}; \vec{q}]$ and it is immediate to see that this implies that

$$\vec{p} \models \varphi; \vec{q}$$

as desired. □

We conclude with a simple corollary which gives a slightly different view on the minimality result. The corollary makes an explicit comparison of the labels of symbolic transitions, showing that those in a STS generated by *Prog*(PC) are as weak as possible in the sense formalised below.

For any formula φ of the logic $\mathsf{SL}_{\mathsf{PC}}$ let $[\![\varphi]\!] = \{(p, \vec{q}) : p \models \varphi; \vec{q}\}$. Then we say that φ is *weaker* than ψ when $[\![\varphi]\!] \supseteq [\![\psi]\!]$. Such notion extends in the obvious component-wise way to tuples of formulae.

The next definition is intended to formalise the intuitive idea of a symbolic transition subsuming another one.

Definition 36 (mimicking symbolic transitions) *Let $C[\vec{X}] \xrightarrow{\vec{\varphi}}_a D[\vec{Y}]$ be a symbolic transition generated by Prog(PC) with a refutation d. We say that such transition is* mimicked *by a correct symbolic transition $C[\vec{X}] \xrightarrow{\vec{\varphi'}}_a D[\vec{Y}]$ if for all tuples of components $\vec{p}$ and $\vec{q}$ such that $\vec{p} \models \vec{\varphi'}; \vec{q}$, we have $C[\vec{p}] \to_a D[\vec{q}]$ with a refutation d' such that $\mathsf{Tr}(d) \preceq \mathsf{Tr}(d')$.*

Then the following corollary holds.

Corollary 37 *Let $C[\vec{X}] \xrightarrow{\vec{\varphi}}_a D[\vec{Y}]$ be a symbolic transition generated by Prog(PC). Then for any correct symbolic transition $C[\vec{X}] \xrightarrow{\vec{\varphi'}}_a D[\vec{Y}]$ which mimics the original one $\vec{\varphi}$ is weaker than $\vec{\varphi'}$.*

Proof. Let $C[\vec{X}] \xrightarrow{\vec{\varphi}}_a D[\vec{Y}]$ be a symbolic transition generated by *Prog*(PC) with a refutation d and assume that it is mimicked by the correct transition $C[\vec{X}] \xrightarrow{\vec{\varphi'}}_a D[\vec{Y}]$.

Let $(\vec{p}, \vec{q}) \in [\![\vec{\varphi'}]\!]$, i.e., $\vec{p} \models \varphi'; \vec{q}$. Then by definition, we have $C[\vec{p}] \to_a D[\vec{q}]$ with a refutation d' such that $\mathsf{Tr}(d) \preceq \mathsf{Tr}(d')$.

Then, by Proposition 35, there is $\vec{q'}$ such that $D[\vec{q'}] = D[\vec{q}]$ and $\vec{p} \models \vec{\varphi}; \vec{q'}$, i.e., $(\vec{p}, \vec{q'}) \in [\![\vec{\varphi}]\!]$. Note that by the first equality, $\vec{q'} = \vec{q}$ and thus $(\vec{p}, \vec{q}) \in [\![\vec{\varphi}]\!]$. Therefore $[\![\vec{\psi}]\!] \subseteq [\![\vec{\varphi}]\!]$ and thus $\vec{\varphi}$ is weaker than $\vec{\psi}$ as desired. □

6 Extending the methodology

In this section we analyse the case of calculi with structural axioms, which are dealt with by using non-syntactical forms of unification, and we discuss the case of calculi in *positive GSOS* format. These results are intended to suggest the extensibility of the proposed approach, while an exhaustive discussion about its applicability to the various existing formats is beyond the scope of this paper.

6.1 Algebraic process calculi with structural axioms

Consider a process calculus PC with a (non-empty) set of structural axioms E, inducing a structural congruence $\equiv$ over coordinators and components. We will show that under mild conditions over the theory E, the unification-based approach to the construction of the STS can be generalised to this case.

According to [4], given two coordinators $C[\vec{X}]$ and $D[\vec{Y}]$ an *E-unifier* is a substitution θ such that $C[\vec{X}];\theta \equiv D[\vec{Y}];\theta$. A *complete set of most general E-unifiers* for $C[\vec{X}]$ and $D[\vec{Y}]$ is a set U of E-unifiers such that for any E-unifier γ there exist $\theta \in U$ and a substitution δ satisfying $\gamma = \theta;\delta$. Then we say that E-unification is *finitary* (*infinitary*) if all unifiable $C[\vec{X}]$ and $D[\vec{Y}]$ admit a finite (infinite) complete set of most general E-unifiers. It is called *nullary* if a complete set of most general E-unifiers does not exist in general.

Let us call E-Prolog an idealised Prolog interpreter which uses E-unification instead of syntactic unification (see, e.g., [28] for the idea of combining logic programming and equational unification). When the set of most general E-unifiers contains more than one substitution, the interpreter chooses in a (don't know) non-deterministic fashion an element in the set.

Consider a program $\mathit{Prog}_E(\mathsf{PC})$ obtained from the program $\mathit{Prog}(\mathsf{PC})$ given in Definition 30 by replacing the first clause (1) with

```
trs(X,A,Y) :- var(X), X=box(A,Y), !.          (13)
```

This is needed to ensure that such a clause will be exploited only when the source coordinator is a variable. Keeping the old clause (1) this would have not been true. E.g., if the the theory were $E = \{X \equiv 0 \mid X\}$ then clause (1) could be used for solving the goal `?- trs(X|Y, a, Z)` since $\mathtt{X} \mid \mathtt{Y}$ unifies with $\mathtt{box(A,Z)}$. This would lead to the answer $\mathtt{X} = \diamond\mathtt{a}\,\mathtt{X}'$ and $\mathtt{Y} = \mathtt{0}$, which would not be the most general one allowing for the step to be performed.

Then we can generalise Theorem 31 to this setting.

THEOREM 38 *Let E be a set of structural axioms such that E-unification is not nullary. Then the* STS *specified by $\mathit{Prog}_E(\mathsf{PC})$, executed by an E-Prolog interpreter, is sound and complete.*

Proof. (*Sketch.*) The proof is essentially the same as for Theorem 31. It can be trivially adapted by replacing syntactical equality with structural congruence $\equiv$. In particular, observe that soundness only relies on the fact that, at each step, the Prolog interpreter computes an E-unifier between a (sub)goal and the head of a clause. Completeness uses the fact that the considered unifiers are complete sets of most general unifiers, so that, as it happens in the absence of structural axioms, any step of a ground refutation will arise as the instance of a refutation step done for the corresponding coordinator. $\square$

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

trs(A.X|‘A, tau, X).

trs(X|Y, tau, X|Z) :- trs(Y, tau, Z).
```

Figure 5: The prolog program for the calculus *asCCS*.

Observe that given a coordinator $C[\vec{X}]$, the number of answers to the query `?- trs(C[X], a, Y)` will be finite if the number of proof rules of the process calculus is finite and E-unification is finitary.

EXAMPLE 39 As an example let us consider a simple asynchronous CCS-like calculus *asCCS*, with a *parallel composition operator* "|", subject to AC1 axioms, i.e. associativity, commutativity and identity

$$(X \mid Y) \mid Z \equiv X \mid (Y \mid Z) \qquad X \mid Y \equiv Y \mid X \qquad X \mid 0 \equiv X$$

where 0 is a distinguished component in the calculus. Moreover, the parallel composition operator allows a single component to move autonomously, performing an action that is reflected at topmost level, i.e., the proof rules for the parallel composition include

$$\frac{X \rightarrow_{\tau} X'}{X \mid Y \rightarrow_{\tau} X' \mid Y} \ (par)$$

and, in addition, only one rule for asynchronous communication

$$\frac{}{a.\, X \mid \bar{a} \rightarrow_{\tau} X} \ (comm)$$

Note that this rule imposes that the communicating processes are close to each other, a situation which could require the use of AC1 axioms to be reached. The Prolog program $Prog_{AC1}(asCCS)$ can be found in Figure 5, where `‘a` is the program representation for $\bar{a}$.

Consider the coordinator $C[X] = a.0 \mid X \mid a.0$, and the goal

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

Then the computation proceeds as follows. First, the third clause of $Prog_{AC1}(asCCS)$ can be used. Take a copy of the clause with fresh variables

```
trs(X1|X2, A1, Z1|X2) :- trs(X1, A1, Z1).
```

The set of most general AC1-unifiers between the head of the clause and the goal includes three substitutions θ_i ($i \in \{1, 2, 3\}$) such that $\theta_i(\mathtt{X}) = \mathtt{X3|X4}$, $\theta_i(\mathtt{A1}) = \mathtt{tau}$, $\theta_i(\mathtt{Z}) = \mathtt{Z1|X2}$ and

$$\begin{array}{ll} \theta_1(\mathtt{X1}) = \mathtt{X3|a.0}, & \theta_1(\mathtt{X2}) = \mathtt{X4|a.0} \\ \theta_2(\mathtt{X1}) = \mathtt{X3}, & \theta_2(\mathtt{X2}) = \mathtt{X4|a.0|a.0} \\ \theta_3(\mathtt{X1}) = \mathtt{X3|a.0|a.0}, & \theta_3(\mathtt{X2}) = \mathtt{X4} \end{array}$$

Assume that substitution θ_1 is chosen. Then `X=X3|X4`, `Z=Z1|X2=Z1|X4|a.0` and the goal is reduced to

```
?- trs(X3|a.0, tau, Z1).
```

Then the second clause of $Prog_{AC1}(asCCS)$ can be used. Take a copy of the clause with fresh variables

```
trs(A2.W|'A2, tau, W).
```

It is easy to see that there exists a unique most general AC1-unifier η between the head of the clause and the goal

$$\eta(\mathtt{W}) = \mathtt{0},\ \eta(\mathtt{X3}) = \mathtt{'a},\ \eta(\mathtt{A2}) = \mathtt{a},\ \eta(\mathtt{Z1}) = \mathtt{0}$$

and this concludes the refutation, with computed answer substitution `X='a|X2` and `Z=X2|a.0`, corresponding to a symbolic transition

$$C[X] \xrightarrow{\bar{a}|Y}_{\tau} Y \mid a.0 \tag{14}$$

It is worth observing that the idea of considering coordinators up to structural congruence and taking syntactical unification would not work. More precisely, the STS in which the symbolic transitions of each coordinator $C[\vec{X}]$ are obtained by considering any $C'[\vec{X}] \equiv C[\vec{X}]$ and taking the symbolic transitions produced for $C'[\vec{X}]$ by a standard Prolog interpreter, with syntactical unification, is not complete. As an example, it is easy to verify that for

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

we would get the answers `X = 'a`, `Z = 0|a.0` and `X = box(tau,Y)`, `Z = a.0|Y|a.0` corresponding to the symbolic transitions

$$C[X] \xrightarrow{\bar{a}}_{\tau} a.\,0 \mid 0 \qquad C[X] \xrightarrow{\diamond\tau.\,Y}_{\tau} a.\,0 \mid Y \mid a.\,0$$

However this does not fully capture the behaviour of the coordinator, i.e. the resulting STS is not complete. Indeed, the first transition is less general than (14), obtained by resorting to AC1-unification. More explicitly, replacing X by $\bar{a} \mid \bar{a}$ we get the transition

$$C[\bar{a} \mid \bar{a}] \rightarrow_{\tau} a.\,0 \mid 0 \mid \bar{a}$$

which would not have any symbolic counterpart. □

Now, while the extension to process calculi with structural axioms and the consequent use of E-unification allow the theoretical framework to be generalised quite smoothly, it is worth observing that from a computational perspective some further issues have to be considered. One is obviously the complexity of E-unification which, depending on the theory E at hand, can be intractable or even undecidable (see [4, 27]). Additionally, as a consequence of the use of E-unification, it can happen that some computations of the E-Prolog interpreter are infinite, when we should simply get as answer "no". This means that, as shown in Theorem 38, the STS defined by $Prog_E(\mathsf{PC})$ is complete, in the sense that any ground transition is the instance of a symbolic transition produced by an E-Prolog computation. However, by the presence of infinite computation, it could be problematic to compute all the symbolic transitions of a given coordinator, using a `findall` construct.

As an example, consider again the calculus $asCCS$ with AC1 parallel composition and the same goal as in Example 39

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

If in the first step of the refutation we still consider the same clause as before, but we use the most general unifier θ_3 we reduce to

```
?- trs(X3|a.0|a.0, tau, Z).
```

i.e., to the starting goal, leading to a potentially infinite reduction.

It is easy to see that the problem is related to rule (*par*) which involves only variables and the AC1 operator "|". Thus, due to the presence of a neutral element 0 for parallel composition, the application of the corresponding clause does not necessarily reduces the goal to structurally less complex ones.

While a general treatment of this problem goes beyond the scope of this paper, we discuss the typical case of an algebraic calculi with AC1 parallel composition, including (*par*) in the proof rules. In this case, a simple solution to avoid such non-termination consists of modifying the proof rules of the calculus, according to a transformation that does not modify its semantics. This approach follows a set of proposals developed in order to address the study of rewriting and deduction systems in presence of equational theories, like the AC1 equivalence axioms (see, e.g., [29]).

The problematic rule (*par*) is removed and any proof rule r of the calculus where "|" occurs in the left-hand side of the conclusion as topmost operator is replaced by a new rule r', obtained by r by adding a parallel generic component which stays idle in the transition (and plays the role of accumulation variables in AC1 rewriting), i.e. any rule r of the kind

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

is replaced by a new rule r'

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

Clearly the new proof rules, like r', are "valid" in the original proof system, and due to the presence of the neutral element, they subsume the original rule r. Hence the transformation does not affect the semantics of the calculus.

Now, it is easy to see that with the new set of rules, for any coordinator $C[\vec{X}]$, each computation starting from the goal

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

is finite. In fact, if we define the complexity of a goal `?- trs(C[X1, ..., Xn], A, D[Y1, ..., Ym])` as the number of operators, different from |, occurring in `C[X1, ..., Cn]`, at any step the subgoals produced by the use of a clause will have a strictly smaller complexity than the original one.

6.2 Symbolic bisimulation: examples on ACCS

After having presented STSs and a constructive methodology for their definition, we now discuss some examples based on the calculus ACCS, with structural axioms, defined in Example 1.

We refer to the "canonical" STS, proved sound and complete, determined by the Prolog construction presented above. As discussed in the previous Section 6.1, if A denotes the AC1 axioms for the parallel operator, then the program $Prog_A(\mathsf{ACCS})$ contains

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

as the first clause and the straightforward translation of the other proof rules of ACCS. Moreover, in order to avoid trivial non-termination, we follow the construction described at the end of Section 6.1, i.e. the clause corresponding to rule (*par*) is removed, and rule (*open*), which has the parallel as topmost operator, gives rise to the clause

```
trs(amb(N,P) | open(N,Q) | R, tau, P|Q|R).
```

Additionally, recall that unification is performed up to structural axioms. Note that some rules in Figure 2 introduce the need of spatial formulae (their left-hand sides require specific composite structures), while the rules (*par*), (*comm*) and (*amb*) call for behavioural modalities, since their premises refer to observable actions.

To have a grasp of the properties of the calculus, let us consider the two components $n[a.0 \mid \bar{a}.0]$ and $m[b.0 \mid \bar{b}.0]$, which are ambients with different names and internal actions. Both components 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. On the other hand, it is easy to note that names distinguish ambients so that, for instance, bisimilarity fails to be a congruence for this calculus, e.g. 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 $m[b.0 \mid \bar{b}.0]$).

Taking into consideration coordinators, the components $n[a.0 \mid \bar{a}.0]$ and $m[b.0 \mid \bar{b}.0]$ can be seen as (bisimilar) instances of $n[X]$ and $m[X]$. It is easy to verify $n[X] \not\sim_{\mathrm{s}} m[X]$, in fact, due to rule (*out*):

$$n[X] \quad \xrightarrow{Y|m[out\ n.\ Z|W]}_{\tau} \quad n[Y] \mid m[Z \mid W],$$

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

$$m[X] \quad \xrightarrow{Y|n[out\ m.\ Z|W]}_{\tau} \quad m[Y] \mid n[Z \mid W].$$

Actually, $n[X] \not\sim_{\mathrm{u}} m[X]$, since they are distinguished by $X = k[out\ n.0]$, and hence, by Theorem 21, $n[X] \not\dot{\sim}_{\mathrm{l}} m[X]$.

An example of coordinators related by $\sim_{\mathrm{s}}$, and hence, because of Theorems 15 and 21, also by $\dot{\sim}_{\mathrm{l}}$ and $\sim$, is: $n[m[out\ n.X]] \sim_{\mathrm{s}} n[0] \mid m[a \mid \bar{a}.X]$. In fact, the two coordinators have the only symbolic transitions below, respectively, which lead to obviously bisimilar coordinators:

$$n[m[out\ n.X]] \quad \xrightarrow{Y}_{\tau} \quad n[0] \mid m[Y] \qquad \text{and} \qquad n[0] \mid m[a \mid \bar{a}.X] \quad \xrightarrow{Y}_{\tau} \quad n[0] \mid m[Y].$$

Not surprisingly, $n[m[out\ n.X]]$ and $n[0] \mid m[a \mid \bar{a}.\tau.X]$ are weak bisimilar. Indeed, as before

$$n[m[out\ n.X]] \quad \xrightarrow{Y}_{\tau} \quad n[0] \mid m[Y] \qquad \text{and} \qquad n[0] \mid m[a \mid \bar{a}.\tau.X] \quad \xrightarrow{Y}_{\tau} \quad n[0] \mid m[\tau.Y]$$

and, trivially, $n[0] \mid m[Y] \ \approx_{\mathrm{w}} \ n[0] \mid m[\tau.Y]$.

Another example of coordinators related by weak symbolic bisimilarity but not equivalent according to strict symbolic bisimilarity is

$$m[n[\mathit{out}\ m.a.b.X]] \mid \mathit{open}\ n.\bar{a}.0 \ \approx_{\mathrm{w}} \ n[m[0]] \mid b.\mathit{open}\ n.X$$

Note that in this case the weak symbolic bisimilarity identifies coordinators that are able to exhibit the same observable action b, although they are quite different structurally.

6.3 Positive GSOS process calculi

The *positive GSOS* format (GSOS) [8], see Figure 4(c), requires the source of the conclusion of any rule to be of the kind $f(X_1, ..., X_n)$, where $f \in \Sigma_n$ is a single operator and, differently from ALG, not a generic context. On the other hand, it allows more general premises where each argument X_i, whose index is in $I \subseteq \{1, \ldots, n\}$, can be tested more than once. The variables Z_i occurring in the target $D[\vec{Z}]$ of the conclusion of the rule are a subset of $\{X_i \mid 1 \leq i \leq n\} \cup \{Y_{i,j} \mid i \in I \wedge 1 \leq j \leq m_i\}$. The targets $Y_{i,j}$ of tested arguments and, additionally, tested variables themselves can occur in $D[\vec{Z}]$ (i.e., it is possible to test an argument without letting it move). All the variables are distinct. It this setting coordinators are not assumed to be linear in their variables.

EXAMPLE 40 An example of GSOS calculus is given by FCCS, the CCS fragment of ACCS (including the rules *pref*, *com* and *par*, see Figure 2) extended with an operator $\mathit{fork}(\cdot)$ and with the additional rule

$$\frac{P \ \rightarrow_{\alpha} \ Q_1 \quad P \ \rightarrow_{\alpha} \ Q_2}{\mathit{fork}(P) \ \rightarrow_{\alpha} \ Q_1 \mid Q_2} \ (\mathit{fork})$$

□

The Prolog program $\mathit{Prog}_{\mathrm{G}}(\mathsf{PC})$ for a GSOS process calculus PC includes, as in the case of ALG (Definition 30), the translation of the SOS rules into Horn clauses:

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

where $\{i_1, ..., i_k\}$ is the set of indexes I of the corresponding proof rule. Each `Zh`, with $1 \leq h \leq n$, can either be `Yij` or `Xi`, for given i, j according to the GSOS rule structure. Each variable `Xi` may occur several times within `Xi1, ..., Xik` and the corresponding `Yij`s (possibly occurring within `Z1,...,Zn`) represent the residuals obtained by the multiple tests for `Xi`. This fact might lead to impose multiple (behavioural) constraints on the same component, in order to allow a transition to happen. This require to extend the chosen STS logic $\mathsf{SL}_{\mathsf{PC}}$ with a conjunction operator "$\wedge$".

The clause (1) of the program in Definition 30 becomes inadequate for GSOS calculi. In fact, instead of simply recording into the variable `Xi` the unique corresponding formula and residual, it is necessary to accumulate all the formulae and residuals relative to multiple tests for `Xi`. The corresponding GSOS clause uses the cut operator, as in Definition 30, and a distinct term `and(_)` to accumulate `box(A,Y)` terms in a list:

```
trs(and(X),A,Y) :- last(X,box(A,Y)), !.                (15)
```

More precisely, this clause, the first one of the program, can unify with a goal whose first element is either a variable or it has already been instantiated with `and([...])`. Here, `last(L, E)` is defined so as to force `L` to be a list where `E` is added as the last but one element and a fresh variable is kept as last element. This allows information to be appended in the list when other premises for the same variable `Xi`, now instantiated to `and([...,box(a,Y), _])`, are considered. This can be implemented as follows:

```
last([Y|_],E) :- var(Y), Y = E, !.
last([_|Z],E) :- last(Z,E).
```

The first clause is selected when the first argument in a query, say X, unifies with a list whose head is a variable followed by any non-empty tail (note that in our program X is initially a variable that will be forced to have such a structure, and this guarantees that the tail is, in turn, a variable). The second clause is selected to recursively scan a list until the condition stated by the first clause is reached.

Each STS transition $C[\vec{X}] \xrightarrow{\varphi}_{\alpha} D[\vec{Y}]$ is determined from the computed answer substitution θ for `trs(C[X1,...,Xn],A,Z)` as follows:

(1.g) if `Xi`$;\theta =$ `and([box(a1, Y1),...,box(ak, Yk), Yk+1])` then the φ_i component of the trigger φ will be $\diamond a_1 \, Y_1 \ \wedge \ldots \ \wedge \ \diamond a_k \, Y_k \ \wedge \ Y_{k+1}$.

(2.g) $D[\vec{Y}]$ is the computed answer substitution for `Z`. The possibility of multiply testing a variable, say `Xi`, and also letting it occur unchanged in the target after a test, may lead to occurrences of `Xi`$;\theta =$ `and([box(a1, Y1),...,box(ak, Yk), Yk+1])` in the computed answer substitution for `Z`, i.e. in `D[`$\vec{\mathtt{Y}}$`]`. Each such occurrence is replaced by the variable `Yk+1`, which, appearing as a conjunct in the formula for `Xi`, represents the continuation of the unchanged component (see next examples).

EXAMPLE 41 Consider the above introduced FCCS calculus and the coordinator $\mathit{fork}(a.X \mid Y)$. Amongst the computed answer substitutions returned for the query `?- trs(fork(a.X|Y),a,Z)` we have:

1. `X=_1, Y=_2, Z= _1|_2 | _1|_2`, and
2. `X=_1, Y=and([box(a,_7),_5]), Z=_1|and([box(a,_7),_5]) | a._1 | _7`

The first one determines the symbolic transition

$$\mathit{fork}(a.X \mid Y) \xrightarrow{X_1, Y_1}_{a} X_1 \mid Y_1 \mid X_1 \mid Y_1$$

while the second leads to

$$\mathit{fork}(a.X \mid Y) \xrightarrow{X_1, \diamond a \, Y_1 \wedge Y_2}_{a} X_1 \mid Y_2 \mid a.X_1 \mid Y_1$$

Note that in this case the target coordinator is obtained from the substitution for `Z` by replacing the term `and([box(a,_7),_5])`, representing an untested occurrence of Y, with variable Y_2 as explained in (2.g) above. □

EXAMPLE 42 Consider the FCCS calculus extended with the next GSOS rule and the corresponding Prolog clause:

$$\frac{P \;\rightarrow_\alpha\; Q}{do(P) \;\rightarrow_\alpha\; \alpha \mid P}\ (do)$$

```
trs(do(X), a, a | X) :- trs(X, a, Y).
```

Given the query `?- trs(do(X),a,Z)`, the computed answer substitution `X = and([box(a,_1)],_2), Z = a | and[box(a,_1),_2]` defines the symbolic transition

$$do(X) \overset{\diamond a.Y \wedge X_1}{\longrightarrow}_a a \mid X_1$$

$\square$

The result in Theorem 31 can be extended to the setting of GSOS calculi, showing that $Prog_{\mathrm{G}}(\mathsf{PC})$, as defined above, generates a correct and complete STS. The proof is complicated by the possible occurrence of the same variable in more goals, breaking the independence of the corresponding refutations. For the sake of simplicity, the proof has been decomposed in some lemmata.

First, we characterise the substitutions, corresponding to STS formulae, determined by a $Prog_{\mathrm{G}}(\mathsf{PC})$ program. A GSOS substitution for $\vec{X}$ associates to each variable in $\vec{X}$ either a variable or a list $\mathtt{and}([\mathtt{box}(\mathtt{a1},\mathtt{Y1}),\ldots,\mathtt{box}(\mathtt{ak},\mathtt{Yk}),\mathtt{\_n}])$ to be interpreted as a set of behavioural constraints, whose order is immaterial.

DEFINITION 43 (GSOS SUBSTITUTION) *A substitution θ is a* GSOS substitution *over $\vec{\mathtt{X}}$ if for each $\mathtt{X_i}$ either $\mathtt{Xi};\theta = \mathtt{Yj}$ or $\mathtt{Xi};\theta = \mathtt{and}([\mathtt{box}(\mathtt{a1},\mathtt{Y1}),\ldots,\mathtt{box}(\mathtt{ak},\mathtt{Yk}),\mathtt{\_n}])$. For any $\mathtt{X_i}$ we define $\theta(\!|X_i|\!) = \emptyset$ if $\mathtt{Xi};\theta = \mathtt{Yj}$ and $\theta(\!|X_i|\!) = \{\mathtt{a1},\ldots,\mathtt{ak}\}$, otherwise.*

LEMMA 44 *Given a coordinator $C[\vec{X}]$, each computed answer substitution θ for a query $?-\ \mathtt{trs}(\mathtt{C}[\vec{\mathtt{X}}],\mathtt{a},\mathtt{Z})$ is a* GSOS *substitution over $\vec{\mathtt{X}}$.*

Proof. By induction on the length of the refutation. $\square$

We next define an order on substitutions. Intuitively speaking, a substitution is smaller than another one if it is more general, i.e., if the corresponding formula is weaker.

DEFINITION 45 (ORDER ON GSOS SUBSTITUTIONS) *Let η, η_1, …, η_n be* GSOS *substitutions over $\vec{X}$. Then we define $\eta \sqsubseteq \{\eta_1,\ldots,\eta_n\}$ over $\vec{\mathtt{X}}$ if for each $\mathtt{Xi}$ we have*

$$\eta(\!|\mathtt{Xi}|\!) \subseteq \eta_1(\!|\mathtt{Xi}|\!) \cup \ldots \cup \eta_n(\!|\mathtt{Xi}|\!)$$

We will write $\eta \sqsubseteq \eta_1$ for $\eta \sqsubseteq \{\eta_1\}$.

Roughly, we have $\eta \sqsubseteq \{\eta_1,\ldots,\eta_n\}$ over $\vec{\mathtt{X}}$ when for each $\mathtt{Xi}$, any constraint $\mathtt{box}(\mathtt{a},\mathtt{Y})$ in $\mathtt{Xi};\eta$ appears also in $\mathtt{Xi};\eta_j$, for some j.

The next technical lemma relates GSOS computed answer substitutions for coordinators to those for instances of the same coordinators, and it is needed by the following lemmata. It relies on the capability implemented by the `last/2` predicate of merging substitutions for the same variable.

LEMMA 46 *Let $C_1[\vec{X}],\ldots,C_m[\vec{X}]$ be coordinators and let ζ be a* GSOS *substitution over $\vec{\mathtt{X}}$. Then,*

1. *if a computed answer substitution θ exists for*

$$?-\texttt{trs}(\texttt{C1}[\vec{\texttt{X}};\zeta],\texttt{a1},\texttt{Z1}),\ldots,\texttt{trs}(\texttt{Cm}[\vec{\texttt{X}};\zeta],\texttt{am},\texttt{Zm}).$$

such that $\zeta;\theta$ is a GSOS *substitution over $\vec{\texttt{X}}$, then a* GSOS *computed answer substitution η over $\vec{\texttt{X}}$ such that $\eta \sqsubseteq \zeta;\theta$ over $\vec{\texttt{X}}$, exists for*

$$?-\texttt{trs}(\texttt{C1}[\vec{\texttt{X}}],\texttt{a1},\texttt{Z1}),\ldots,\texttt{trs}(\texttt{Cm}[\vec{\texttt{X}}],\texttt{am},\texttt{Zm}).$$

2. *if a* GSOS *computed answer substitution η over $\vec{\texttt{X}}$, exists for*

$$?-\texttt{trs}(\texttt{C1}[\vec{\texttt{X}}],\texttt{a1},\texttt{Z1}),\ldots,\texttt{trs}(\texttt{Cm}[\vec{\texttt{X}}],\texttt{am},\texttt{Zm}).$$

then a computed answer substitution θ, such that $\zeta;\theta \sqsubseteq \{\zeta,\eta\}$ and $\zeta;\theta$ is a GSOS *substitution over $\vec{\texttt{X}}$, exists for*

$$?-\texttt{trs}(\texttt{C1}[\vec{\texttt{X}};\zeta],\texttt{a1},\texttt{Z1}),\ldots,\texttt{trs}(\texttt{Cm}[\vec{\texttt{X}};\zeta],\texttt{am},\texttt{Zm}).$$

Proof. See [6] (by induction on the length of refutations and exploiting the definition of the `last/2` predicate).

We can now prove two other lemmata, which will be used in the proofs of soundness and completeness, respectively. In the sequel we will denote by $\vec{\texttt{X}}_{\texttt{j}}$ the variables that occur in $C_j[\vec{X}]$.

LEMMA 47 *Let $C_1[\vec{X}],\ldots,C_k[\vec{X}]$ be coordinators. If a $Prog_{\mathrm{G}}(\mathsf{PC})$ refutation for the query*

$$?-\texttt{trs}(\texttt{C1}[\vec{\texttt{X}}],\texttt{a1},\texttt{U1}),\ldots,\texttt{trs}(\texttt{Ck}[\vec{\texttt{X}}],\texttt{ak},\texttt{Uk}).$$

exists with a GSOS *computed answer substitution θ, then a refutation for*

$$?-\texttt{trs}(\texttt{Cj}[\vec{\texttt{X}}_{\texttt{j}}],\texttt{aj},\texttt{Uj}).$$

exists, for any $1 \le j \le k$, with a GSOS *computed answer substitution η such that $\eta \sqsubseteq \theta$ over $\vec{\texttt{X}}_{\texttt{j}}$. Moreover $\texttt{Uj};\eta^* = \texttt{Uj};\theta^*$, where η^* and θ^* are obtained from η and θ by applying the transformation described in (2.g) above.*

Proof. See [6] (by Lemma 46 (1)).

LEMMA 48 *Let $C_1[\vec{X}],\ldots,C_m[\vec{X}]$ be coordinators. If a $Prog_{\mathrm{G}}(\mathsf{PC})$ refutation for*

$$?-\texttt{trs}(\texttt{Cj}[\vec{\texttt{X}}],\texttt{aj},\texttt{Uj}).$$

exists for each $1 \le j \le k$ with a GSOS *computed answer substitution η_j, then a refutation for*

$$?-\texttt{trs}(\texttt{C1}[\vec{\texttt{X}}],\texttt{a1},\texttt{U1}),\ldots,\texttt{trs}(\texttt{Ck}[\vec{\texttt{X}}],\texttt{ak},\texttt{Uk}). \tag{16}$$

exists with a GSOS *computed answer substitution θ, such that $\theta \sqsubseteq \{\eta_1,\ldots,\eta_k\}$ over $\vec{\texttt{X}}$. Moreover $\texttt{Uj};\eta^*_{\texttt{j}} = \texttt{Uj};\theta^*$, where η^*_j and θ^* are obtained from η_j and θ by using the transformation described in (2.g) above.*

Proof. See [6] (by Lemma 46 (2)).

Ultimately, the theorem about soundness and completeness for the STS generated by the Prolog program for GSOS calculi can be proved.

THEOREM 49 *If* PC *is in* GSOS *format,* $Prog_G$(PC) *defines a sound and complete* STS.

Proof. The proof follows the same structure of that of Theorem 31, the main differences being induced by non-linearity of variables in the premises of the rules (which is dealt by Lemma 47 for soundness and Lemma 48 for completeness).

Soundness. It is necessary to prove that given a symbolic transition of the STS

$$C[X_1,\ldots,X_n] \stackrel{\varphi_1,\ldots,\varphi_n}{\longrightarrow}_a D[Y_1,\ldots,Y_h] \tag{17}$$

for any $\vec{p},\vec{q}$ such that $\vec{p} \models \vec{\varphi};\vec{q}$ a derivation $C[p_1,\ldots,p_n] \rightarrow_a D[q_1,\ldots,q_h]$ exists.

Recall that the above transition is obtained from a successful refutation of the query

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

with computed answer substitution `Xi =` φ_i^+, and `Z = D[Y1, ..., Yh]`, and each φ_i is obtained following the steps (1.g)-(2.g) above described. The proof is by induction on z, the length of such refutation.

- ($z = 1$) This case, where the matching clause has empty body, works exactly as for the corresponding case of Theorem 31.
- ($z > 1$) A successful refutation for a coordinator $C[\vec{X}] = f(C_1[\vec{X}],\ldots,C_l[\vec{X}])$ starts by unifying the above query with the head of a clause, say

```
trs(f(W1,...,Wl), a, F[Z1,...,Zm]) :-  trs(Wi1, a1, U1),...,
                                        trs(Wik, ak, Uk).
```

  where $I = \{i_1,\ldots,i_k\}$ is the set of the indexes in the premise of the corresponding proof rule. We recall that the same variable may occur more times within {`Wi1, ..., Wik`} and each `Zh` can either be `Wi` or `Uj`, for some `i` and `j`. Let $H \subseteq I$ denote the set of indexes h such that `Zh = Uj` for some j (i.e. `Zh` is the continuation of a tested variable). Unification gives a most general unifier ψ such that for each $1 \leq i \leq l$, `Wi`$;\psi =$ $\mathtt{C_i}[\vec{\mathtt{X}}]$ and `Z`$;\psi =$ `F[Z1,...,Zm]`, i.e. `Wij`$;\vec{\psi}$ are coordinators. By hypothesis, a successful refutation for

  ?- `trs(Wi1;`$\vec{\psi}$`, a1, U1),...,trs(Wik;`$\vec{\psi}$`, ak, Uk).`

  exists, with computed answer substitution $\vec{\xi}$. We indicate with $\vec{\theta}$ the restriction of $\vec{\xi}$ relative to the variables occurring in `Wi1`$;\vec{\psi}$ `...Wik`$;\vec{\psi}$, i.e. $\vec{\mathtt{X}}$, and with $\vec{\eta}$ the one relative to `U1,..., Uk`. Formulae in $\vec{\theta}$ may represent a conjunction of constraints over multiply tested variables. We indicate with $\vec{\eta^*}$ the substitution obtained from $\vec{\eta}$ by replacing any `and([...])` with a fresh variable, with $\vec{\theta^*}$ the substitution obtained from $\vec{\theta}$ by suitably adding these fresh variables as conjuncts, as described in (2.g) above. By the rule format, $\vec{\eta^*}$ is purely spatial. The union of $\vec{\theta^*}$ and $\vec{\eta^*}$ is indicated as $\vec{\xi^*}$. Consequently, in (17)

$$\varphi_i = X_i;\vec{\psi};\vec{\xi^*} = X_i;\vec{\theta^*}$$

$$D[\vec{Y}] = F[\vec{Z}];\vec{\psi};\vec{\xi^*} = F[\vec{Z};\vec{\psi};\vec{\xi^*}] \text{ where } Z_h;\vec{\psi};\vec{\xi^*} = \begin{cases} W_i;\vec{\psi} & \text{if} \quad h \notin H \\ U_{ij};\vec{\eta^*} & \text{if} \quad h \in H \end{cases}$$

since the formulae $Z_h;\vec{\psi};\vec{\xi^*}$ are purely spatial.

By Lemma 47, a successful refutation for any query

$$\texttt{?- trs(Wij;}\vec{\psi}\texttt{, aj, Uj).}$$

exists, with computed answer substitutions $\vec{\mu}_j$ for the variables in $\texttt{Wij};\vec{\psi}$ and ν_j for $\texttt{Uj}$. It holds that $\vec{\mu}_j \sqsubseteq \vec{\theta}$ over $\vec{X}_j$ and $\texttt{U}_\texttt{j};\nu^*_\texttt{j} = \texttt{U}_\texttt{j};\eta^*_\texttt{j}$, where $\vec{X}_j$ are the variables in $X_1,\ldots,X_k$ occurring in $\texttt{Wij};\vec{\psi}$. This successful refutation determines the symbolic transition

$$W_{ij};\vec{\psi} \xrightarrow{\vec{\mu^*_j}}_{a_j} U_j;\eta^*_j$$

Since the length of every such refutation is clearly less than z, by inductive hypothesis the above transition is sound and $\forall \vec{u},\vec{v}$ such that $\vec{u} \models \vec{\mu^*}_{J_j};\vec{v}$ the ground transition

$$W_{ij};\vec{\psi};\vec{u} \rightarrow_{a_j} U_j;\eta^*_j;\vec{v} \tag{18}$$

exists for any $j \in I$. Each one of these transitions separately satisfies one premise of the proof rule corresponding to the first selected Prolog clause.

We now show that each $\vec{p},\vec{q}$ such that $\vec{p} \models \vec{\phi}^*;\vec{q}$ guarantee that all the premises are satisfied and hence the ground transition can take place, i.e. for suitable $\vec{r},\vec{s}$

$$C[p_1,\ldots,p_n] = f(r_1,\ldots,r_l) \rightarrow_a F[s_1,\ldots,s_m] = D[q_1,\ldots,q_h].$$

Since $\vec{p} \models \vec{X};\vec{\psi};\vec{\xi}^*;\vec{q}$ and the formulae $\vec{X};\vec{\psi}$ are purely spatial, then the residual $\vec{t}$ of $\vec{p}$ after satisfying $\vec{X};\vec{\psi}$ is uniquely determined, and $\vec{t} \models \vec{\theta}^*;\vec{q}$. Then, by associativity and being $\vec{\psi}$ an unifier:

$$\begin{aligned} C[\vec{p}] &= C[\vec{X};\vec{\psi};\vec{t}] \\ &= C[\vec{X}];\vec{\psi};\vec{t} \\ &= f(\vec{W});\vec{\psi};\vec{t} \\ &= f(\vec{W};\vec{\psi};\vec{t}) \end{aligned}$$

Let be $\vec{r} = \vec{W};\vec{\psi};\vec{t}$, then $\vec{r} \models \vec{W};\vec{\psi};\vec{\theta}^*;\vec{q}$. For $j \in I$, let us write the components of $\vec{r}$ as $r_j = W_j;\vec{\psi};\vec{f_j}$, where the components of $\vec{f_j}$ are determined as the residual of r_j after satisfying $W_j;\vec{\psi}$.[1] Then, $\vec{f_j} \models \theta^*_j;\vec{q}$, i.e. $\vec{f} \models \vec{\theta}^*;\vec{q}$. From $\mu_j \sqsubseteq \theta$, it follows that for each $j \in I$ $f_j \models \mu^*_j;\vec{q}$, i.e. all the premises (18) are satisfied by $\vec{r}$ and the transition

$$C[p_1,\ldots,p_n] = f(r_1,\ldots,r_l) \rightarrow_a F[s_1,\ldots,s_m]$$

occurs. According to the format of the selected clause (note that all the formulae here are spatial, i.e. can be read as coordinators/components)

$$s_j = \begin{cases} r_j = Z_j;\vec{\psi};\vec{\xi}^*;\vec{q} & \text{if} \quad j \notin I \\ U_j;\eta^*_j;\vec{q} = Z_j;\vec{\psi};\vec{\xi}^*;\vec{q} & \text{if} \quad j \in I \end{cases}$$

[1] Note that differently from the case of ALG, an r_i component can be tested more times. Accordingly, being $W_j\psi = C_j[\vec{X}]$, the components of $\vec{\theta}^*_j$ may consist of the conjunction of the constraints imposed (over $\vec{X}$) by the different tests. Each component of $\vec{f_j}$ satisfies the corresponding conjunction.

(Note that, in order to guarantee that satisfaction with residuals works properly, s_j is defined for each $j \in I$, although only s_j with $j \in H$ actually occurs in the target of the transition).

Finally, $F[s_1, \ldots, s_m] = F[\vec{Z}; \vec{\psi}; \vec{\xi}^*; \vec{q}] = F[\vec{Z}; \vec{\psi}; \vec{\xi}^*]; \vec{q} = D[\vec{Y}]; \vec{q} = D[\vec{q}]$ follows.

Completeness. It is necessary to prove that for any coordinator $C[\vec{X}]$, a tuple of components $\vec{p}$ and a component q such that $C[\vec{p}] \rightarrow_a q$, the goal

$$? - \mathtt{trs}(\mathtt{C}[\vec{\mathtt{X}}], \mathtt{a}, \mathtt{Z}). \tag{19}$$

returns a computed answer substitution $\vec{X} = \vec{\varphi^+}$ and $Z = \vec{\eta}^+$ such that $\exists\ \vec{r}$ with $\vec{p} \models \vec{\varphi}; \vec{r}$ and $Z; \vec{\eta}; \vec{r} = q$, where $\vec{\varphi}$ and $\vec{\eta}$ are obtained from $\vec{\varphi^+}$ and $\vec{\eta}^+$ following the steps (1.g)-(2.g) above described. The existence of $C[\vec{p}] \rightarrow_a q$ implies the existence of a $Prog_{\mathrm{G}}(\mathsf{PC})$ refutation for

$$\mathtt{?\text{-}\ trs(C[}\vec{p}\mathtt{]\ ,\ a,\ q).}$$

(which does not use clause (15)). Let us proceed by induction on the length z of such refutation.

- ($z = 1$) As for the corresponding premise-less cases of Theorem 31 (possibly managing the transformation (1.g)-(2.g) of computed answer substitutions).
- ($z > 1$) The case $C[\vec{X}] = X$ is resolved by the modal clause, as in the corresponding point of Theorem 31. Otherwise, let

  $$\mathtt{trs}(\mathtt{f}(\vec{\mathtt{W}}), \mathtt{a}, \mathtt{F}[\vec{\mathtt{Z}}]) : -\mathtt{trs}(\mathtt{Wi1}, \mathtt{a1}, \mathtt{Z1}), \ldots, \mathtt{trs}(\mathtt{Wik}, \mathtt{ak}, \mathtt{Zk}). \tag{20}$$

  be the first clause used in the refutation of $\mathtt{trs}(\mathtt{C}[\vec{\mathtt{p}}], \mathtt{a}, \mathtt{q})$. The unification of the query with the clause yields a most general unifier of $\mathtt{C}[\vec{\mathtt{X}}]; \mathtt{p}$ and $\mathtt{f}[\vec{\mathtt{W}}]$, which consists, as in the soundness part of this proof, of the purely spatial substitution $\vec{\psi}$ for $\vec{\mathtt{W}}$, and a tuple of components $\vec{\mathtt{s}}$ such that

  $$\mathtt{C}[\vec{\mathtt{p}}] = \mathtt{f}[\vec{\psi}; \vec{\mathtt{p}}] \qquad \mathtt{C}[\vec{\mathtt{X}}] = \mathtt{f}[\vec{\psi}] \qquad \mathtt{F}[\vec{\mathtt{s}}] = \mathtt{q}.$$

  The original ground goal is thus reduced to the k ground subgoals:

  $$\mathtt{trs(Wi1;}\vec{\psi};\vec{p}\mathtt{,\ a1,\ }s_1\mathtt{)}, \ldots, \mathtt{\ trs(Wik;}\vec{\psi};\vec{p}\mathtt{,\ ak,\ }s_k\mathtt{).}$$

  obtained by instantiating the body of the clause. Note that for some l, j it may happen that $\mathtt{Wil;}\vec{\psi}; \vec{p} = \mathtt{Wij;}\vec{\psi}; \vec{p}$. The existence of a refutation for the conjunction of the ground subgoals guarantees that a $Prog_{\mathrm{G}}(\mathsf{PC})$ refutation for each of them exists, and it is clearly shorter than z. Each one of these refutations corresponds to the ground transition $\mathtt{Wij}; \vec{\psi}; \vec{\mathtt{p}} \rightarrow_{\mathtt{aj}} \mathtt{s_j}$, which occurs as a premise in the first proof rule used to derive the ground transition $C[p] \rightarrow_a q$. By inductive hypothesis, a refutation for each

  $$\mathtt{trs}(\mathtt{Wij}; \vec{\psi}, \mathtt{aj}, \mathtt{Zj})$$

  exists with computed answer substitution $\vec{\gamma}_j^+$ for the variables in $\mathtt{Wij}; \vec{\psi}$ and ζ_j^+ for $\mathtt{Zj}$. This defines the symbolic transition (with $\vec{\gamma}_j, \zeta_j$ obtained as usual according to $(1.g) - (2.g)$ above)

$$W_{ij};\vec{\psi} \xrightarrow{\vec{\gamma}_j}_{a_i} \mathtt{Z_j}\zeta_j$$

such that, for any j there exists a tuple of components $\vec{u}_j$ satisfying $\vec{p}_j \models \vec{\gamma}_j;\vec{u}_j$, where $\vec{p}_j$ is a suitable subset of the components of $\vec{p}$, i.e. those that according to the rule occur in the $j-th$ premise, and $Z_j;\zeta_j;\vec{u}_j = s_j$.

By Lemma 48, a refutation for

$$\mathtt{trs}(\mathtt{Wi1};\vec{\psi},\mathtt{a1},\mathtt{Z1}),\ldots,\mathtt{trs}(\mathtt{Wik};\vec{\psi},\mathtt{ak},\mathtt{Zk})$$

exists with computed answer substitution $\vec{\gamma}$, over the variables in $\vec{W}\vec{\psi}$, i.e. $\vec{X}$, and $\vec{\zeta}$ (as usual, form $\vec{\zeta}^{+}$) over $\vec{\mathtt{Z}}$ such that $\vec{\gamma} \sqsubseteq \{\vec{\gamma}_1,\ldots,\vec{\gamma}_k\}$, $\vec{p} \models \vec{\gamma}\vec{u}$ and $Z_j;\vec{\zeta} = Z_j;\zeta_j$.

Therefore, the query (19) unifies with the clause (20), determining the symbolic transition

$$C[\vec{X}] \xrightarrow{\vec{\psi};\vec{\gamma}}_a D[\vec{Y}] = F[\vec{Z};\vec{\zeta}]$$

To conclude, recall that $\vec{p} \models \vec{\gamma};\vec{u}$ and $\vec{X};\vec{\psi};\vec{\gamma} = \vec{X};\vec{\gamma}$, and hence $\vec{p} \models \vec{\psi};\vec{\gamma};\vec{u}$, and $D[\vec{u}] = F[\vec{Z};\vec{\zeta};\vec{u}] = F[\vec{s}] = q$, as desired. $\square$

6.4 Further notes on minimality

In Section 5.2 we have discussed to which extent the use of unification may guarantee some property of minimality for the constructed STS's in the case of ALG process calculi. The same issue is discussed here in the case of calculi with structural axioms and GSOS calculi, even though a detailed treatment goes beyond the scope of this paper.

The most relevant difference between ALG and GSOS formats is the fact that for calculi adhering to the latter format the possibility of multiply testing a variable in a rule immediately leads to consider non-linear coordinators. On the other hand, GSOS rules impose simpler constraints on components, i.e. the computed answer substitutions for the variables consist only of modal constraints, which can be easily accumulated during the computation, and give rise to conjunctive formulas for multiply tested variables. This has been exploited in extending the soundness and completeness proof form ALG to GSOS. We conjecture that the same technique can be used to extend the, similar in structure, proof of minimality to GSOS calculi.

The case of ALG calculi with structural equivalence, with the consequent use of unification modulo the equivalence, appears to be more complex because of the possibility of having several most general unifier for a given refutation. In this case, Proposition 35 could be generalised as follows: A fixed refutation d can lead to a set of symbolic transition $C[\vec{X}] \xrightarrow{\vec{\varphi}_i}_a D_i[\vec{Y}_i]$, with $i \in I$. Then, for all components $\vec{p}$, s, if $C[\vec{p}] \to_a s$ is proved by a refutation d' satisfying $\mathsf{Tr}(d) \preceq \mathsf{Tr}(d')$ then there exists an index $i \in I$ such that $s = D_i[\vec{q}]$ for some tuple of components $\vec{q}$ with $\vec{p} \models \vec{\varphi}_i;\vec{q}$. This is scope for future work.

7 Related work

The notion of STS has been influenced by several related formalisms. Symbolic approaches to behavioural equivalences can be found in [26, 39], while the idea of using spatial logic formulae as an elegant mathematical tool for combining structural and behavioural constraints has been separately proposed in [18, 24]. Many different kinds of labelled transition systems for coordinators have been previously proposed in the literature, starting from context systems [32] and structured transition systems [21], till more recent proposals like tile systems [25] and conditional transition systems [38]. The common point is to define abstract equivalences on coordinators by labelling the transitions with trigger-effect pairs, where the triggers express the hypotheses on process variables under which a global transition can be performed. Roughly, the distinguishing feature of our approach w.r.t. all the frameworks above is the greater generality of symbolic labels which account for spatial constraints over unspecified components.

The use of spatial formulae makes our framework tailored to a wider class of calculi than those which essentially rely on modal formulae alone, like [32, 21, 25, 38] above. Interestingly, the research in the above frameworks has led to the definition of convenient specification formats that guarantee properties such as that bisimilarity is a congruence. It would be interesting to develop similar results for our framework, too.

The idea of using unification for building the triggering formulae comes from Logic Programming and more precisely by its view as an interactive system presented in [13].

In case of LTSs with a unique label τ (that can be regarded as reduction semantics), our approach seems to share some analogies with narrowing techniques used in rewrite systems, and it would be interesting to formally compare the two approaches.

Some close relations exist also with the work on *modal transition systems* [31], where both transitions that *must* be performed and transitions which are only *possible* can be specified. Consequently the syntax of the calculus is extended with two kind of prefix operators $\Box a.()$ and $\Diamond a\,()$. We recall also the logical process calculus of [20], which mixes CCS and a form of μ-calculus in order to allow some components of the system to be logically specified. Our process logic exhibits some similarities both with the calculus underlying modal transition systems and with the logical process calculus. However, the purpose of the mentioned formalisms is to provide a loose specification of a system, where some components are characterised by means of logical formulae. Instead, in our case open systems are modelled within the original calculus and the STS fully describes their semantics by using the logic to characterise synthetically their possible transitions.

The issue of avoiding universal closure of coordinators finds its dual formulation in avoiding contextual closure of components, which is a current theme of research in the area of process calculi and reactive systems. Specifically, there are two main scenarios. The first scenario consists of a process calculus for which bisimilarity $\sim$ is not a congruence (on components). One can define 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. But note that in general $\simeq$ is not a bisimulation. The largest congruence which is also a bisimulation is called *dynamic bisimilarity* and is defined by allowing context closure at each bisimulation step [36], with transitions like $p \rightarrow_{C[.]} C[p]$ for any p and $C[.]$. The second scenario concerns reactive systems equipped with reduction semantics. The idea is to synthesise an LTS that respects the original reductions and for which bisimilarity is a congruence. This is done by labelling transition with contexts that catalyse reactions, as there is no other predefined notion of observation. In both scenarios the problem is of course to keep the branching of the transition system as small as possible, still guaranteeing the congruence property.

To avoid universal quantification on contexts, several authors [43, 33, 40, 41, 42, 30, 12, 23] propose a symbolic transition system for components whose labels are the "minimal" contexts necessary to the component for evolving:

$$p \stackrel{C[\_,X_1,...,X_n]}{\longrightarrow} D[X_1, ..., X_n]$$

means that $C[p, p_1, ..., p_n]$ can reduce in one step to $D[p_1, ..., p_n]$ for any $p_1, ..., p_n$, and that C is strictly necessary to perform the step.

The technique, originally proposed in [43] with a purely set-theoretical presentation has been further refined in [33] in categorical terms, by expressing minimality in terms of *relative pushouts* in *pre-categories*. The papers [40, 41, 42] provide a more general and elegant framework based on *groupoidal relative pushouts* and *groupoidal 2-categories*, where the groupoid structure is used to deal with structural axioms. A recent work recasts the groupoidal approach in terms of *double categories* and it also investigates weak equivalences [12]. Finally, some relaxed equivalences, called semi-saturated, have been considered in [9], and it has been shown that the relative pushouts approach can be applied also to graph rewriting systems [23].

All the above categorical approaches are very general, robust and elegant, but except for [43] they are not "constructive", in the sense that the context-labelled transitions are characterised very precisely from the mathematical point of view, but in general it cannot be said how to construct them starting from the operational rules of the calculus. On the contrary, our approach provides a constructive way of defining a tractable symbolic transition system, as explained in Section 5. As a future work, it could be worth investigating the connections with [30], which looks very close to the spirit of our approach.

Regarding the symbolic systems in [43], it seems that they could be complemented and made more efficient by exploiting our technique. In fact, even if there transitions always depart from components, they may lead also to contexts (like D above), over which bisimulation is defined via universal closure. Thus, the problem of universal quantification is just shifted from contexts to components (which is the problem we have addressed).

8 Conclusions

We have presented a semantic framework for open processes, here represented as suitable coordinators that can evolve autonomously or dynamically interact with other coordinators and components. Our approach can be said to be variable-driven instead of name-driven (the distinction is explained in the introduction), because the openness of coordinators is modelled by placeholders and composition corresponds to substitution. The operational semantics of coordinators is expressed by symbolic transition systems, whose labels include spatial and modal formulae that constrain the components to be inserted in the coordinator before interaction can take place. One advantage is that the symbolic transition system is typically more tractable than the universal closure on all components. Furthermore, for open systems, the dynamic feeding of coordinators can be more appropriate than static closure.

On top of the operational semantics, we have discussed how to derive some abstract semantics:

- Strict bisimilarity $\sim_{\mathrm{s}}$ is a straight extension of the standard bisimilarity on labelled transition systems.

- Loose bisimilarity $\dot{\sim}_{\mathrm{l}}$ relaxes the structural constraints imposed by spatial formulae, solving in part the problem of redundant symbolic transitions which might cause $\sim_{\mathrm{s}}$ to distinguish "too much".

- Weak bisimilarity $\approx_{\mathrm{w}}$ is another relaxation of $\sim_{\mathrm{s}}$. It is more appropriate for those calculi that include a silent action τ for internal (non-observable) computations.

For sound and complete STSs, it is shown that $\sim_{\mathrm{s}}$ and $\dot{\sim}_{\mathrm{l}}$ imply $\sim_{\mathrm{u}}$ and that $\approx_{\mathrm{w}}$ implies $\approx_{\mathrm{u}}$ (where $\sim_{\mathrm{u}}$ and $\approx_{\mathrm{u}}$ are defined by universal closure).

Interestingly, in many cases sound and complete symbolic transition systems can be derived automatically just starting from the SOS specification of the process calculus, making our approach constructive. The branching width of the synthesised STS is kept to a minimal extent by exploiting most general unifiers and the cut operator.

The results above constitute just the core theory of our framework. Other issues like congruence properties, trace equivalences, and redundancy freeness of symbolic transitions have been discussed in [7].

Future work will include the study of more complex examples and applications, the treatment of nominal calculi (thus reconciling the variable driven approach with the name-driven approach), and the investigation of specification formats and additional hypotheses that allow symbolic equivalences to most suitably match their corresponding universal closures.

In particular, regarding nominal calculi, we plan to develop the treatment of names and name restriction in order to deal with the modelling of fresh and restricted/secret resources. The idea will be to extend the notion of STS and the underlying process logic so as to deal with a logical notion of freshness, possibly taking inspiration from [15, 16]. The higher-order unification mechanism of λ-Prolog [34] could provide a convenient framework for the construction of the relative STS. Another interesting related work in this respect is [44], where a model checker for the π-calculus has been developed in tabled logic programming by using variables and unification to abstract over classes of fresh and restricted names.

Practical applications of our approach seems to be possible in the field of security and protocol verification in particular. Indeed, process calculi have been traditionally used for verification exploiting symbolic semantics and unification-based approaches for dealing with the infiniteness of the execution environments, typically due to the unconstrained generative power of intruders (see, e.g. [1, 19, 10, 11]). Such similarities are worth being further investigated.

Finally, the Prolog-based construction of STSs can be further pursued in an open and dynamic system engineering perspective [2, 3], in which, for instance, the use of *meta* logic constructs for the programmable definition of transitions and more specific reasoning about the structure of the calculus, or the hypothetical, assumption-based reasoning about formulae, e.g., "under which assumptions the process $P \mid X$ can evolve so as to satisfy a given property?", seems to be of interest.

References

[1] M. Abadi and M. Fiore. Computing symbolic models for verifying cryptographic protocols. In *14th Computer Security Foundations Workshop (CSFW-14)*, pages 160–173. IEEE, Computer Society Press, 2001.

[2] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, 1997.

[3] L.F. Andrade, J.L. Fiadeiro, J. Gouveia, G. Koutsoukos, and M. Wermelinger. Coordination for orchestration. In F. Arbab and C. L. Talcott, editors, *Proceedings of COORDINATION'02*, volume 2315 of *LNCS*, pages 5–13. Springer, 2002.

[4] F. Baader and J. H. Siekmann. Unification theory. In D. M. Gabbay, C. J. Hogger, J. A. Robinson, and J. H. Siekmann, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming (2)*, pages 41–126. Oxford University Press, 1994.

[5] P. Baldan, A. Bracciali, and R. Bruni. Bisimulation by unification. In H. Kirchner and C. Ringeissen, editors, *Proceedings of AMAST '02*, volume 2422 of *LNCS*, pages 254–270. Springer, 2002.

[6] P. Baldan, A. Bracciali, and R. Bruni. A semantic framework for open processes. Technical Report TR-07-09, Department of Computer Science, University of Pisa, 2007. Availavble as `http://compass2.di.unipi.it/TR/Files/TR-07-09.pdf.gz`.

[7] P. Baldan, R. Bruni, and A. Bracciali. Symbolic equivalences for open systems. In C. Priami and P. Quaglia, editors, *Global Computing: IST/FET International Workshop, GC 2004*, volume 3267 of *LNCS*, pages 1–17. Springer, 2005.

[8] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.

[9] F. Bonchi, B. Konig, and U. Montanari. Saturated semantics for reactive systems. In *Proceedings of LICS'06*, pages 69–80. IEEE, 2006.

[10] M. Boreale. Symbolic trace analysis of cryptographic protocols. In F. Orejas, P.G. Spirakis, and J. van Leeuwen, editors, *Proceedings of ICALP'01*, volume 2076, pages 667–681. Springer, 2001.

[11] A. Bracciali, G. Baldi, G. Ferrari, and E. Tuosto. A coordination-based methodology for security protocol verification. In N. Busi, R. Gorrieri, and F. Martinelli, editors, *ENTCS*, volume 121. Elsevier, 2005.

[12] R. Bruni, F. Gadducci, U. Montanari, and P. Sobociński. Deriving weak bisimulation congruences from reduction systems. In M. Abadi and L. de Alfaro, editors, *Proceedings of CONCUR'05*, volume 3653 of *LNCS*, pages 293–307. Springer, 2005.

[13] R. Bruni, U. Montanari, and F. Rossi. An interactive semantics of logic programming. *Theory and Practice of Logic Programming*, 1(6):647–690, 2001.

[14] L. Caires. *A Model for Declarative Programming and Specification with Concurrency and Mobility.* PhD thesis, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 1999.

[15] L. Caires and L. Cardelli. A spatial logic for concurrency (part I). In N. Kobayashi and B. Pierce, editors, *Proceedings of TACS*, volume 2215 of *LNCS*, pages 1–37. Springer, 2001.

[16] L. Caires and L. Cardelli. A spatial logic for concurrency (part ii). In L. Brim, P. Jancar, M. Kretínský, and A. Kucera, editors, *Proceedings of CONCUR'02*, volume 2421 of *LNCS*, pages 209–225. Springer, 2002.

[17] L. Cardelli and A. Gordon. Mobile ambients. In M. Nivat, editor, *Proceedings of FoSSaCS'98*, volume 1378 of *LNCS*, pages 140–155. Springer, 1998.

[18] L. Cardelli and A.D. Gordon. Anytime, anywhere. modal logics for mobile ambients. In *Proceedings of 27th ACM Symposium on Principles of Programming Languages*, pages 365–377. ACM, 2000.

[19] 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 *IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.

[20] R. Cleaveland and G. Lüttgen. A logical process calculus. In U. Nestmann and P. Panangaden, editors, *ENTCS*, volume 68(2). Elsevier, 2002.

[21] A. Corradini and U. Montanari. An algebraic semantics for structured transition systems and its application to logic programs. *Theoretical Computer Science*, 103:51–106, 1992.

[22] R. De Simone. Higher level synchronizing devices in MEIJE–SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[23] H. Ehrig and B. König. Deriving bisimulation congruences in the DPO approach to graph rewriting. In I. Walukiewicz, editor, *Proceedings of FoSSaCS'04*, volume 2987 of *LNCS*, pages 151–166. Springer, 2004.

[24] J.L. Fiadeiro, T. S. E. Maibaum, N. Martí-Oliet, J. Meseguer, and I. Pita. Towards a verification logic for rewriting logic. In *Proceedings of WADT '99*, pages 438–458. Springer, 2000.

[25] F. Gadducci and U. Montanari. The Tile Model. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 133–166. MIT Press, 1998.

[26] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.

[27] A. Herold and J. H. Siekmann. Unification in abelian semigroups. *J. Autom. Reasoning*, 3(3):247–283, 1987.

[28] S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *LNCS*. Springer, 1989.

[29] J. P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15, 1986.

[30] B. Klin, V. Sassone, and P. Sobociński. Labels from reductions: towards a general theory. In J. Fiadeiro and J. Rutten, editors, *Proceedings of CALCO'05*, volume 3629 of *LNCS*, 2005.

[31] K. G. Larsen and B. Thomsen. A modal process logic. In *Proceedings of LICS*, pages 203–210. IEEE, 1988.

[32] K.G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. In M.S. Paterson, editor, *Proceedings of ICALP'90, 17th International Colloquium on Automata, Languages and Programming*, volume 443 of *LNCS*, pages 526–539. Springer, 1990.

[33] J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In C. Palamidessi, editor, *Proceedings of CONCUR'00*, volume 1877 of *LNCS*, pages 243–258. Springer, 2000.

[34] D. Miller and G. Nadathur. Higher-order logic programming. In *Handbook of Logics for Artificial Intelligence and Logic Programming*, pages 499–590. Clarendon Press, 1990.

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

[36] U. Montanari and V. Sassone. Dynamic congruence vs. progressing bisimulation for CCS. *Fundamenta Informaticae*, 16(1):171–199, 1992.

[37] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Computer Science Deapartment, 1981.

[38] A. Rensink. Bisimilarity of open terms. *Information and Computation*, 156(1-2):345–385, 2000.

[39] D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33(1):69–97, 1996.

[40] V. Sassone and P. Sobocinski. Deriving bisimulation congruences using 2-categories. *Nordic Journal of Computing*, 10(2):163–183, 2003.

[41] V. Sassone and P. Sobocinski. Locating reaction with 2-categories. *Theor. Comput. Sci.*, 333(1-2):297–327, 2005.

[42] V. Sassone and P. Sobocinski. Reactive systems over cospans. In *Proceedings of LICS'05*. IEEE, 2005.

[43] P. Sewell. From rewrite rules to bisimulation congruences. In D. Sangiorgi and R. de Simone, editors, *Proceedings of CONCUR'98*, volume 1466 of *LNCS*, pages 269–284. Springer, 1998.

[44] P. Yang, C.R. Ramakrishnan, and S.A. Smolka. A logical encoding of the pi-calculus: Model checking mobile processes using tabled resolution. In L.D. Zuck et al., editor, *Proceedings of VMCAI'03*, volume 2575 of *LNCS*, pages 86–101. Springer, 2003.