Symbolic Equivalences for Open Systems*

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

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

Abstract. Behavioural equivalences on open systems are usually defined by comparing system behaviour in all environments. Due to this "universal" quantification over the possible hosting environments, such equivalences are often difficult to check in a direct way. Here, working in the setting of process calculi, we introduce a hierarchy of behavioural equivalences for open systems, building on a previously defined symbolic approach. The hierarchy comprises both branching, bisimulation-based, and non-branching, trace-based, equivalences. Symbolic equivalences are amenable to effective analysis techniques (e.g., the symbolic transition system is finitely branching under mild assumptions), which result to be sound, but often not complete due to redundant information. Two kinds of redundancy, syntactic and semantic, are discussed and and one class of symbolic equivalences is identified that deals satisfactorily with syntactic redundant transitions, which are a primary source of incompleteness.

1 Introduction

The widespread diffusion of web applications and mobile devices has shifted the attention to *open systems*, i.e., systems where mobile software components can be dynamically connected to interact with each other. As a consequence, language-independent frameworks to reason about open systems and software architectures for coordination have gained interest. In the literature, process calculi (PC) have been devised as a useful paradigm for the specification and analysis of open systems. Situated between real programming languages and mere mathematical abstractions, they facilitate rigorous system analysis, offering the basis for prototypical implementation and for verification tools. Indeed, many running implementations exist of languages based on calculi originally proposed to experiment basic interaction primitives [29, 33, 16, 11].

The operational and abstract semantics of PC, as well as algorithms for verification, are often naturally defined for *components*, i.e. closed terms of the calculus, via a labelled transition system (LTS). The extension to *coordinators*, i.e. contexts with holes representing the openness of the system, can require

* Research supported by the Projects IST-2001-32747 AGILE, IST-2001-32617 MYTHS and IST-2001-32530 SOCS.

non-trivial enhancements. Here, in the style, e.g., of the Security Process Algebra (see [19] and references therein), we use process variables as place-holders for unspecified components which may join the system, i.e., coordinators are viewed as terms with process variables. A way to lift a semantic equivalence $\approx$ from components to coordinators is to define $C[X] \approx_{\text{univ}} D[X]$ when $C[p] \approx D[p]$ for all components p. This universal quantification can be recasted in a coalgebraic framework by enriching the original transition system over components: all coordinators $C[X]$ are taken as states and a transition $C[X] \rightarrow_p C[p]$ is added for any component p. However the extended transition system is likely to be intractable, being infinitely branching even for trivial calculi.

In [5], we introduced *symbolic transition systems* (STSs) to ease the analysis of coordinators' properties. An STS is a transition system where states are coordinators and transition labels are logic formulae expressing structural and behavioural requirements on the unknown components which would allow the transition to occur. Symbolic transition systems account for the operational semantics of coordinators, and, based on this, two abstract semantics are defined: *strict bisimilarity* $\sim_{\text{strict}}$, a straight extension of the standard bisimilarity on labelled transition systems, and *large bisimilarity* $\dot{\sim}_{\text{large}}$, introduced as a mean to solve, at least in part, the problem of redundant symbolic transitions (see below) which may cause $\sim_{\text{strict}}$ to distinguish "too much". For suitable STSs (i.e., sound and complete w.r.t. the original LTS) both "symbolic" bisimilarities approximate $\sim_{\text{univ}}$, the standard extension of bisimilarity $\sim$ to coordinators defined by universal quantification, as illustrated above. Moreover, sound and complete STSs can be automatically derived from SOS specifications whenever the SOS rules satisfy rather general syntactic formats.

The first part of this paper consolidates and extends the theory of symbolic bisimilarities. More specifically, we investigate some basic properties of $\sim_{\text{strict}}$ and $\dot{\sim}_{\text{large}}$, showing, e.g., that the latter approximates $\sim_{\text{univ}}$ strictly better than $\sim_{\text{strict}}$, although in general it is non-transitive (incidentally, the dot on top of $\sim$ in the symbol for large bisimilarity is a reminder of this fact). We discuss how the defined equivalences are influenced by *redundant* symbolic specifications, identifying two kinds of redundancy, called *syntactic* and *semantic*. While $\sim_{\text{strict}}$ cannot overcome redundancy at all, $\dot{\sim}_{\text{large}}$ can deal with significant forms of both kinds of redundancy, but it does not fully solve any of the two. This motivates the introduction of a novel bisimilarity $\dot{\sim}_{\text{irred}}$, called *irredundant*, which solves the problem of syntactic redundancy. In general $\dot{\sim}_{\text{large}}$ and $\dot{\sim}_{\text{irred}}$ are not comparable and a fourth, better approximation of $\sim_{\text{univ}}$ can be obtained by combining $\dot{\sim}_{\text{large}}$ and $\dot{\sim}_{\text{irred}}$, originating the "diamond" of bisimilarities in Fig. 1(a).

The second part of the paper fully generalises the STS approach to the setting of *trace semantics*. Albeit trace semantics are usually easier to deal with, in the case of coordinators the problem of universal closure w.r.t. all components still persists and thus also trace equivalences benefit from a symbolic approach. To every kind of symbolic bisimilarity described above, there corresponds a notion of symbolic trace semantics. Each trace semantics is refined by the corresponding bisimilarity, as expected, and all are correct approximations of the universal trace

$$\begin{tikzcd}
 & \sim_{\mathrm{univ}} & \\
 & (\dot\sim_{\mathrm{large}} \cup \dot\sim_{\mathrm{irred}})^* \arrow[u] & \\
\dot\sim_{\mathrm{large}} \arrow[ur] & & \dot\sim_{\mathrm{irred}} \arrow[ul] \\
 & \sim_{\mathrm{strict}} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

(a) Diamond of bisimilarities.

$$\begin{tikzcd}
\simeq_{\mathrm{univ}} & & \dot\simeq_{\mathrm{itight}} \arrow[ll] & \\
 & \dot\simeq_{\mathrm{ltight}} \arrow[ul] & & \simeq_{\mathrm{stight}} \arrow[ul] \arrow[ll] \\
 & & \dot\simeq_{\mathrm{irred}} \arrow[uu] \arrow[uull, bend left] & \\
 & \dot\simeq_{\mathrm{large}} \arrow[uu] \arrow[uuul, bend left] & & \simeq_{\mathrm{strict}} \arrow[uu] \arrow[ll] \arrow[ul] \\
\sim_{\mathrm{univ}} \arrow[uuuu] & & \dot\sim_{\mathrm{irred}} \arrow[ll] \arrow[uu] & \\
 & \dot\sim_{\mathrm{large}} \arrow[ul] \arrow[uu] & & \sim_{\mathrm{strict}} \arrow[ll] \arrow[ul] \arrow[uu]
\end{tikzcd}$$

(b) Tower of symbolic semantics.

Fig. 1.

equivalence $\simeq_{\mathrm{univ}}$. Finally, we introduce a compact form of trace, called *tight*, which is exploited to improve the precision of all the approximations of $\simeq_{\mathrm{univ}}$. Though symbolic trace equivalences are the natural counterparts of symbolic bisimilarities, the notion of *tight trace* is original and fully exploits the use of formulae as transition labels. The full hierarchy of equivalences is in Fig. 1(b).

Synopsis. § 2 recalls the principles of STSs from [5]. All material in § 3–5 is original to this contribution. Relations between $\sim_{\mathrm{strict}}$ and $\dot\sim_{\mathrm{large}}$ are investigated in § 3, while § 4 discusses syntactic and semantic redundancy. § 5 provides a symbolic approach to trace semantics. Technical results come together with examples, based on calculi designed ad-hoc to clarify the features of interest. Some concluding remarks and an account of related work are in § 6.

2 Approximating the Universal Bisimilarity

We restrict here to (non-empty) process calculi based on unsorted signatures. Given a process signature Σ and a denumerable set of variables $\mathcal{X}$ (disjoint from Σ), $\mathbb{T}_\Sigma(\mathcal{X})$ denotes the term-algebra over Σ with variables in $\mathcal{X}$. For $P \in \mathbb{T}_\Sigma(\mathcal{X})$, $var(P)$ denotes the set of variables in P. If $var(P) = \emptyset$ then P is *closed*. Closed terms form the set $\mathcal{P}$ of *components* p (possibly taken modulo a structural congruence $\equiv$), while terms in $\mathbb{T}_\Sigma(\mathcal{X})$ form the set $\mathcal{C}$ of *coordinators* C. With $C[X_1, \ldots, X_n]$ we mean that C is a coordinator such that $var(C) \subseteq \{X_1, \ldots, X_n\}$. To simplify the notation hereafter we shall use single-holed coordinators, i.e., coordinators with at most one variable, but all definitions and results straightforwardly extend to many-holed coordinators.

The operational semantics of process calculi is given in terms of *labelled transition systems* (LTSs). A transition from p to q with observable $a \in \Lambda$ (the label alphabet) is indicated as $p \rightarrow_a q$. Transitions are often specified by a collection of inductive rules, following the SOS paradigm [31]. Throughout the paper PC denotes a fixed process calculus over a signature Σ, with an LTS $\mathcal{L}$ specified by a set of SOS rules.

A *bisimulation* is a symmetric relation $\approx$ over components such that if $p \approx q$, then for any transition $p \rightarrow_a p'$ there exist a component q' and a transition $q \rightarrow_a q'$ with $p' \approx q'$. *Bisimilarity* $\sim$ is the largest bisimulation. The *universal bisimilarity* $\sim_{\text{univ}}$ is the lifting of $\sim$ to coordinators obtained by closing under all substitutions, i.e. $C[X] \sim_{\text{univ}} D[X] \overset{\text{def}}{\iff} \forall p \in \mathcal{P},\ C[p] \sim D[p]$. Since components are closed, $p \sim_{\text{univ}} q$ iff $p \sim q$.

Symbolic Bisimulation. The equivalence $\sim_{\text{univ}}$ can be quite intractable. To address this problem we exploit a symbolic approach based on:

1. abstracting from components not playing an active role in the transition;
2. specifying the active components as little as possible;
3. making assumptions on the structure and behaviour of active components.

The idea is to derive from the LTS a *symbolic transition system* (STS), where states are coordinators and labels are formulae expressing behavioural and structural conditions required to unknown components for enabling the transition.

The logic L_{PC} that we consider has *modal* and *spatial* operators in the style of [8, 12]. 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 considered spatial. The syntax of L_{PC}-formulae φ and the associated notion of satisfaction are given below, where $X \in \mathcal{X}$ denotes a process variable, $f \in \Sigma$ is an operator in the process signature and $a \in \Lambda$ an action label.

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

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

We denote by $var(\varphi)$ the set of variables in a formula φ. We consider *linear* formulae only (i.e. formulae where no variable occurs twice). A formula in L_{PC} is called *spatial* if it only contains variables and spatial operators $f \in \Sigma$ (abusing the notation, spatial expressions can be read both as formulae and as coordinators). Each component p can be regarded as a spatial formula with no variables, and $p \models q$ iff $p \equiv q$.

For instance, 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. the formulae $r.X$ and $\diamond r.X$ in Example 1), we remark that their meaning is quite different: the former imposes a spatial constraint, the latter imposes a behavioural constraint, satisfied by components which can perform the action a (e.g., the process $(b.0 \mid a.0)\backslash b$ in a CCS-like calculus).

Definition 1 (sts). *A symbolic transition system (*STS*) $\mathcal{S}$ for PC is a set of transitions $C[X] \xrightarrow{\varphi}_a D[Y]$ where $C[X]$ and $D[Y]$ are coordinators in PC, $a \in \Lambda$ and φ is a formula in L_{PC} with $var(\varphi) \supseteq var(D)$.*

The correspondence between the variable X in the source and its residual Y in the target is expressed by the occurrence of Y in φ. For example, a symbolic transition in a CCS-like calculus could be $X\backslash b \mathrel{-\{\diamond a.Y\}\!\!\rightarrow_a} Y\backslash b$ for $a \neq b$. The modal formula $\diamond a.Y$ is satisfied by any process p which can perform an action a becoming a generic process, say q. Hence the symbolic transition represents the infinitely many concrete transitions $p\backslash b \rightarrow_a q\backslash b$ which are obtained by replacing X and Y by such p and q, respectively.

To provide an adequate representation of the original transition system, an STS is required to satisfy suitable correspondence properties. Informally, $C[X] \mathrel{-\{\varphi\}\!\!\rightarrow_a} D[Y]$ means that the coordinator C, instantiated with any component p satisfying φ, i.e., $p \models \varphi[q/Y]$, must be able to perform the action a becoming an instance of D, namely $D[q]$. Also, any concrete transition on components should have symbolic counterparts.

Definition 2 (Sound/Complete sts). *An* STS $\mathcal{S}$ *for* PC *is:*

- sound*, if for any symbolic transition* $C[X] \mathrel{-\{\varphi\}\!\!\rightarrow_a} D[Y]$ *in* $\mathcal{S}$ *and for any* p, q *with* $p \models \varphi[q/Y]$*, there exists a transition* $C[p] \rightarrow_a D[q]$ *in the* LTS *of* PC.
- complete*, if for any coordinator* $C[X]$*, for any* p *and for any transition* $C[p] \rightarrow_a r$ *in* PC *there are* q *and* $C[X] \mathrel{-\{\varphi\}\!\!\rightarrow_a} D[Y]$ *in* $\mathcal{S}$ *with* $p \models \varphi[q/Y]$ *and* $r \equiv D[q]$.

Observe that a weaker notion of completeness, simply asking that for any $p \rightarrow_a q$ there exist $C[X] \mathrel{-\{\varphi\}\!\!\rightarrow_a} D[Y]$ and p', q' such that $C[p'] \equiv p$, $D[q'] \equiv q$ and $p' \models \varphi[q'/Y]$ would be inappropriate since a complete STS would not represent the proper computational behaviour of coordinators. For instance, it is easy to see that the LTS of components (seen as a trivial STS) would be complete according to the weaker notion of completeness, although it does not include any transition for terms with variables.

The straightforward definition of bisimulation equivalence over an STS is given below.

Definition 3 ($\sim_{\text{strict}}$). *A symmetric relation* $\approx$ *on coordinators is a* strict symbolic bisimulation *if for all* $C[X]$*,* $D[X]$ *with* $C[X] \approx D[X]$ *and for any symbolic transition* $C[X] \mathrel{-\{\varphi\}\!\!\rightarrow_a} C'[Y]$*, there exists* $D[X] \mathrel{-\{\varphi\}\!\!\rightarrow_a} D'[Y]$ *such that* $C'[Y] \approx D'[Y]$*. The largest strict symbolic bisimulation* $\sim_{\text{strict}}$ *is an equivalence called* strict symbolic bisimilarity

Strict bisimilarity requires a transition to be simulated by a transition with exactly the same label. 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 easy to check can be exploited in symbolic bisimilarity (e.g., to standardise the labels) and the theory easily carries over.

Strict symbolic bisimilarity distinguishes at least as much as universal bisimilarity, i.e. $\sim_{\text{strict}}$ implies $\sim_{\text{univ}}$ (Theorem 1 below, taken from [5]), but the converse does not hold in general. A better approximation of $\sim_{\text{univ}}$ is obtained by relaxing the requirement of exact (spatial) matching between formulae.

Definition 4 (**$\dot\sim_{\text{large}}$**)**.** *A symmetric relation $\approx$ on coordinators is a* large symbolic bisimulation *if for all $C[X]$, $D[X]$ with $C[X] \approx D[X]$ and for any transition $C[X] \xrightarrow{\varphi}_{a} C'[Y]$ there exist a transition $D[X] \xrightarrow{\psi}_{a} D'[Z]$ and a spatial formula η such that $\varphi = \psi[\eta/Z]$ and $C'[Y] \approx D'[\eta]$. The greatest large bisimulation $\dot\sim_{\text{large}}$ is called* large symbolic bisimilarity*.*

As a trivial example, let $\Sigma = \{a, f(.), g(.)\}$ and take the STS with transitions $f(X) \xrightarrow{Y}_{\tau} Y$, $g(X) \xrightarrow{Y}_{\tau} Y$, and $g(X) \xrightarrow{a}_{\tau} a$. Obviously, $f(X) \not\sim_{\text{strict}} g(X)$, because $f(X)$ cannot match the last transition of $g(X)$, while $f(X) \dot\sim_{\text{large}} g(X)$ since the formula X is "more general" than the spatial formula a.

Theorem 1 (**$\sim_{\text{strict}} \Rightarrow \dot\sim_{\text{large}} \Rightarrow \sim_{\text{univ}}$**)**.** *For any sound and complete* STS *and for all coordinators $C[X]$, $D[X]$ we have*

$$C[X] \sim_{\text{strict}} D[X] \quad \Rightarrow \quad C[X] \dot\sim_{\text{large}} D[X] \quad \Rightarrow \quad C[X] \sim_{\text{univ}} D[X].$$

Bisimulation by Unification. The framework introduced in [5] is completed by a constructive definition of a suitable STS associated to any PC whose operational proof rules are in *algebraic format* [20] (that generalises, e.g., the well-known De Simone format [17]). Starting from the algebraic SOS proof rules for PC, a Prolog program $Prog_{\mathrm{A}}(\mathsf{PC})$ can be derived which specifies a *sound* and *complete* STS over L_{PC}. The program defines a predicate `trs(X,A,Y)` whose intended meaning is "any component satisfying X can perform a transition labelled by A and become a component satisfying Y". Then, given a coordinator $C[X]$, if the query `?- trs(C[X], A, Z)` is successful, the corresponding computed answer substitution represents a symbolic transition for the coordinator. The code in $Prog_{\mathrm{A}}(\mathsf{PC})$ consists of the obvious translation of the SOS rules into Horn clauses, with an additional rule to handle behavioural formulae. Intuitively, the unification mechanism is used to compute the minimal requirements on the process variables of a coordinator which allow an SOS rule to be applied. We remark that if the set of SOS rules of PC is finite, then the program $Prog_{\mathrm{A}}(\mathsf{PC})$ has a finite number of clauses and the defined STS is finitely branching (even if the whole STS has instead infinitely many states and transitions, as obviously it must include all the original transitions over components).

3 Properties of Strict and Large Bisimilarities

In this section we study some basic properties of $\sim_{\text{strict}}$ and $\dot\sim_{\text{large}}$, and we show, by means of a few examples, that they capture different notions of simulation.

Comparing $\sim_{\text{strict}}$ and $\dot\sim_{\text{large}}$. The relation $\dot\sim_{\text{large}}$ is always coarser than $\sim_{\text{strict}}$. On the other hand, $\dot\sim_{\text{large}}$ is *not* guaranteed to be an equivalence relation, since it may fail to be transitive in some "pathological" situations (as the one below).

Example 1. Consider the simple process calculus SC, whose processes $P \in \mathcal{P}$ are:

$$P \ ::= \ 0 \mid r.P \mid l(P) \mid k_1(P) \mid k_2(P) \mid k_3(P)$$

$$\begin{array}{ccc}
k_1(X) \dashrightarrow_{\{l(r.Y)\}} {}_o k_1(Y) & k_2(X) \dashrightarrow_{\{l(r.Y)\}} {}_o k_2(Y) & k_3(X) \dashrightarrow_{\{l(r.Y)\}} {}_o k_3(Y) \\
k_1(l(X)) \dashrightarrow_{\{\diamond r.Y\}} {}_o k_1(Y) & k_2(l(X)) \dashrightarrow_{\{\diamond r.Y\}} {}_o k_2(Y) & k_3(l(X)) \dashrightarrow_{\{\diamond r.Y\}} {}_o k_3(Y) \\
 & k_2(l(X)) \dashrightarrow_{\{r.Y\}} {}_o k_2(Y) & k_3(X) \dashrightarrow_{\{l(r.l(Y))\}} {}_o k_3(l(Y))
\end{array}$$

Fig. 2. Symbolic transitions for $k_i(X)$ and $k_i(l(X))$

$$k_2(X) \begin{array}{c}\sim_{\text{strict}} \\ \dot\sim_{\text{large}} \\ \sim_{\text{univ}}\end{array} k_1(X) \begin{array}{c}\not\sim_{\text{strict}} \\ \dot\sim_{\text{large}} \\ \sim_{\text{univ}}\end{array} k_3(X) \begin{array}{c}\not\sim_{\text{strict}} \\ \dot{\not\sim}_{\text{large}} \\ \sim_{\text{univ}}\end{array} k_2(X)$$

$$k_2(l(X)) \begin{array}{c}\not\sim_{\text{strict}} \\ \dot{\not\sim}_{\text{large}} \\ \sim_{\text{univ}}\end{array} k_3(l(X))$$

Fig. 3. Example 1 illustrated

This calculus is not intended to represent a meaningful case study, but just a way to illustrate some peculiarities of our theory. For mnemonic reasons r can be interpreted as a generic resource, $l(\_)$ as a locking mechanism, and $k_1(\_)$, $k_2(\_)$ and $k_3(\_)$ as three different access keys which may open and fetch a locked resource. The operational semantics of SC is given by the reduction rules below:

$$r.P \rightarrow_r P \qquad\qquad k_i(l(r.P)) \rightarrow_o k_i(P) \quad i = 1, 2, 3$$

The use of a *resource* r is represented by a transition labelled by r, while the use of a key to unlock a process generates a transition labelled by o (*open*).

Let $\mathcal{S}$ be any sound and complete STS whose transitions for the open terms $k_1(X)$, $k_2(X)$, $k_3(X)$, $k_1(l(X))$, $k_2(l(X))$, $k_3(l(X))$ are exactly the ones in Fig. 2 (for instance, they could have been generated by separate specifications provided for each k_i by different system analysts). Observe that in $\mathcal{S}$:

- $k_1(X) \dot\sim_{\text{large}} k_3(X)$, since $k_3(X) \dashrightarrow_{\{l(r.l(Y))\}} {}_o k_3(l(Y))$ can be simulated by the instance of $k_1(X) \dashrightarrow_{\{l(r.Z)\}} {}_o k_1(Z)$, where Z is replaced by the spatial formula $l(Y)$. Note that, instead, $k_1(X) \not\sim_{\text{strict}} k_3(X)$.
- $k_2(X) \dot\sim_{\text{large}} k_1(X)$, because $k_2(X) \sim_{\text{strict}} k_1(X)$.
- $k_2(X) \dot{\not\sim}_{\text{large}} k_3(X)$ since $k_3(X) \dashrightarrow_{\{l(r.l(Y))\}} {}_o k_3(l(Y))$ cannot be simulated via $k_2(X) \dashrightarrow_{\{l(r.Z)\}} {}_o k_2(Z)$ with Z replaced by $l(Y)$, as the target processes $k_3(l(Y))$ and $k_2(l(Y))$ are not large bisimilar. In fact, though, in this specific example, the formulae $\diamond r.Y$ and $r.Y$ are satisfied by the same components (i.e., $\{r.p \mid p \in \mathcal{P}\}$), the moves of coordinators $k_3(l(Y))$ and $k_2(l(Y))$, respectively labelled by $\diamond r.Y$ and $r.Y$, cannot be related in the (strict or large) bisimulation game. However it holds that $k_3(l(Y)) \sim_{\text{univ}} k_2(l(Y))$.

The outcome of the above discussion is summarised in Fig. 3. In particular, it shows that $\dot\sim_{\text{large}}$ is non-transitive, i.e., $\dot\sim_{\text{large}} \circ \dot\sim_{\text{large}} \not\subseteq \dot\sim_{\text{large}}$. Moreover, in general, $\sim_{\text{strict}} \subsetneq \dot\sim_{\text{large}} \subsetneq \sim_{\text{univ}}$, i.e., all the inclusions in Theorem 1 are proper.

Therefore, both $\sim_{\text{strict}}$ and $\dot{\sim}_{\text{large}}$ have some pros and cons. In fact $\sim_{\text{strict}}$ is always an equivalence and, in view of an automated verification, its simpler formulation is helpful. Furthermore, being defined as the straightforward notion of bisimilarity on the STS, existing tools and techniques can be reused. On the other hand, $\dot{\sim}_{\text{large}}$ yields a more precise approximation of $\sim_{\text{univ}}$ and, from Theorem 1 it immediately follows that, for sound and complete STSs, $(\dot{\sim}_{\text{large}})^* \Rightarrow \sim_{\text{univ}}$. Hence in using $\dot{\sim}_{\text{large}}$ as a proof technique for $\sim_{\text{univ}}$, transitivity can still be exploited.

Congruence Properties. Call a relation $\cong$ on coordinators an *outer-congruence* if $C[X] \cong D[X]$ implies $C[E[Y]] \cong D[E[Y]]$ for any $E[Y]$, an *inner-congruence* if $C[X] \cong D[X]$ implies $E[C[X]] \cong E[D[X]]$ for any $E[Y]$, and a *quasi-congruence* if it is both an inner- and an outer-congruence. A quasi-congruence which is an equivalence is called a *congruence.* While $\sim_{\text{univ}}$ is an outer-congruence by definition, in general, $\sim_{\text{strict}}$ and $\dot{\sim}_{\text{large}}$ are *not*: taking the calculus SC in Example 1 and the STS $\mathcal{S}$ therein, we have $k_2(X)\dot{\sim}_{\text{large}}k_1(X)$, but $k_2(l(X))\not\dot{\sim}_{\text{large}}k_1(l(X))$. Actually, $k_2(X) \sim_{\text{strict}} k_1(X)$ and thus also $\sim_{\text{strict}}$ is not an outer-congruence. However, since $\sim_{\text{univ}}$ is an outer-congruence and both $\dot{\sim}_{\text{large}}$ and $\sim_{\text{strict}}$ are correct approximations of $\sim_{\text{univ}}$, we can reduce the proof of $C[E[Y]] \sim_{\text{univ}} D[E[Y]]$ to the proof of $C[X]\dot{\sim}_{\text{large}}D[X]$ or $C[X] \sim_{\text{strict}} D[X]$.

Many SOS formats have been introduced to guarantee that bisimilarity is a congruence. This property can be lifted to the symbolic level for PC in De Simone format [17] (a special case of the algebraic format) by taking the STS defined via the Prolog program $Prog_{\text{A}}(\text{SC})$ mentioned in § 2. Moreover, by Definition 4, the absence of spatial operators in the premises of De Simone rules, and hence in the formulae used as labels in the STS, guarantees $\sim_{\text{strict}}= \dot{\sim}_{\text{large}}$.

Proposition 1. *If* PC *is in De Simone format, then* $Prog_{\text{A}}(\mathsf{PC})$ *yields a* STS *where* $\sim_{\text{strict}}$ *is a congruence and* $\sim_{\text{strict}}= \dot{\sim}_{\text{large}}$.

The generalisation of Proposition 1 to other SOS formats (e.g., GSOS) is non-trivial, because they are incomparable w.r.t. the algebraic format and thus $Prog_{\text{A}}(\mathsf{PC})$ (see § 2) cannot be exploited to define a sound and complete STS.

4 Syntactic and Semantic Redundancy

A sound and complete STS may have several different symbolic transitions departing from the same coordinator $C[X]$ but whose instances cover non-disjoint sets of component behaviours. In this section we discuss the influence of redundant symbolic specifications on symbolic bisimilarities. The following example shows that we can distinguish between two kinds of redundancy: syntactic and semantic.

Example 2. Consider the calculus SC in Example 1, where $k_1(X)\dot{\sim}_{\text{large}}k_3(X)$, but $k_1(X) \not\sim_{\text{strict}} k_3(X)$ (see Fig. 3). The equivalence $\sim_{\text{strict}}$ distinguishes the two coordinators because of the symbolic transition $k_3(X) \xrightarrow{\{l(r.l(Y))\}}_o k_3(l(Y))$, which is an instance of the more general transition $k_3(X) \xrightarrow{\{l(r.Y)\}}_o k_3(Y)$. This is what we call *syntactic* redundancy.

On the other hand, $k_2(l(X)) \not\sim_{\text{strict}} k_3(l(X))$ and $k_2(l(X)) \not\dot{\sim}_{\text{large}} k_3(l(X))$, while $k_2(l(X)) \sim_{\text{univ}} k_3(l(X))$. Roughly, this is due to the fact that the two distinct symbolic transitions $k_2(l(X)) \text{-}\{r.Y\}\!\!\rightarrowtail_o k_2(Y)$ and $k_2(l(X)) \text{-}\{\diamond r.Y\}\!\!\rightarrowtail_o k_2(Y)$ characterise the same set of concrete component transitions (since, in SC, the different formulae $r.Y$ and $\diamond r.Y$ are satisfied by the same processes). This is an aspect of what we call *semantic* redundancy (in general more complex cases can arise, whose solution is not as obvious as here and that cannot be recasted simply in terms of formula equivalences).

4.1 Syntactic Redundancy and Irredundant Bisimilarity

For solving syntactic redundancy the idea is to consider a symbolic bisimulation that takes into account only the "more general" symbolic transitions. For simplicity, we consider calculi without structural axioms.

Definition 5 (Irredundant Transition). *Given a coordinator $C[X]$ in an* STS*, a transition $C[X] \text{-}\{\varphi\}\!\!\rightarrowtail_a C'[Y]$ is called* redundant *if there exists a transition $C[X] \text{-}\{\psi\}\!\!\rightarrowtail_a C''[Z]$ and a spatial formula $\chi \neq Y$ such that $C''[\chi] = C'[Y]$, and $\psi[\chi/Z] = \varphi$. A transition is called* irredundant *if it is not redundant.*

In Example 1, the presence of the (irredundant) transition $k_3(X) \text{-}\{l(r.Y)\}\!\!\rightarrowtail_o k_3(Y)$ makes $k_3(X) \text{-}\{l(r.l(Y))\}\!\!\rightarrowtail_o k_3(l(Y))$ a redundant transition.

Definition 6 ($\dot{\sim}_{\text{irred}}$). *A symmetric relation $\approx$ on coordinators is an* irredundant symbolic bisimulation *if for all $C[X]$, $D[X]$ such that $C[X] \approx D[X]$, for any irredundant transition $C[X] \text{-}\{\varphi\}\!\!\rightarrowtail_a C'[Y]$, there is a transition $D[X] \text{-}\{\varphi\}\!\!\rightarrowtail_a D'[Y]$ such that $C'[Y] \approx D'[Y]$. The largest irredundant symbolic bisimulation $\dot{\sim}_{\text{irred}}$ is called* irredundant symbolic bisimilarity*.*

Like large bisimilarity, also $\dot{\sim}_{\text{irred}}$ might fail to be an equivalence (because of the lack of transitivity). However, the syntactical property in Proposition 2, when satisfied by an STS, guarantees transitivity.

Proposition 2. *Let $\mathcal{S}$ be an* STS *such that for any redundant symbolic transition $C[X] \text{-}\{\varphi\}\!\!\rightarrowtail_a C'[Y]$, if $C[X] \text{-}\{\psi\}\!\!\rightarrowtail_a C''[Z]$ and there exists a spatial formula χ with $\psi[\chi/Z] = \varphi$, then $C''[\chi] = C'[Y]$. Then $\dot{\sim}_{\text{irred}}$ is transitive.*

As mentioned above, large and irredundant bisimilarities, although arising from similar motivations, are (in general) not comparable. To see, for instance, that $\dot{\sim}_{\text{irred}} \not\subseteq \dot{\sim}_{\text{large}}$ consider Example 2. Recall that $k_2(X) \not\dot{\sim}_{\text{large}} k_3(X)$, but instead $k_2(X) \dot{\sim}_{\text{irred}} k_3(X)$, since transition $k_3(X) \text{-}\{l(r.l(Y))\}\!\!\rightarrowtail_o k_3(l(Y))$ is redundant and thus $k_2(X)$ and $k_3(X)$ have the "same" irredundant transitions. An analogous counterexample shows that $\dot{\sim}_{\text{large}} \not\subseteq \dot{\sim}_{\text{irred}}$ (see [6]). Hence it can be useful to combine $\dot{\sim}_{\text{large}}$ and $\dot{\sim}_{\text{irred}}$, as, of course, for any sound and complete STS, $(\dot{\sim}_{\text{large}} \cup \dot{\sim}_{\text{irred}})^* \Rightarrow \sim_{\text{univ}}$. The relationships between the bisimilarities introduced so far are summarised in Fig. 1(a), where arrows represent subset inclusion.

Again, the absence of spatial operators in the premises of De Simone rules, and hence in the formulae used as labels in the STS, guarantees $\sim_{\text{strict}} = \dot{\sim}_{\text{irred}}$.

Proposition 3. *If* PC *is in De Simone format, then* $\mathit{Prog}_{\mathrm{A}}(\mathsf{PC})$ *yields a* STS *where* $\sim_{\text{strict}} = \dot\sim_{\text{irred}}$.

4.2 On Semantic Redundancy

The fact that $\dot\sim_{\text{large}}$ and $\dot\sim_{\text{irred}}$ are incomparable shows that large bisimulation goes beyond syntactic redundancy. Large bisimilarity has been introduced to avoid the distinction between two coordinators $C[X]$ and $D[X]$ that can perform the same transitions, apart from transitions which are, in a sense, instances of other existing transitions. However, in practice, since redundancy check is nested inside the definition, $\dot\sim_{\text{large}}$ can deal with a more general notion of redundancy, which has a semantic flavour.

The ideal situation would be when the whole hierarchy in Fig. 1(a) collapses into the simplest symbolic bisimilarity $\sim_{\text{strict}}$, which could then be used as a *complete* proof technique for $\sim_{\text{univ}}$.

However, when sketching the proof of the possible implication $\sim_{\text{univ}} \Rightarrow \sim_{\text{strict}}$, one soon realizes that $\sim_{\text{univ}}$ can hardly be formulated as a strict bisimilarity. In fact assume $C[X] \sim_{\text{univ}} D[X]$, and take any symbolic move $C[X] \xrightarrow{\varphi(Y)}_a C'[Y]$ 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 $C[p_i] \rightarrow_a C'[q_i]$. Then, since $C[X] \sim_{\text{univ}} D[X]$, for any such move, we must have $D[p_i] \rightarrow_a d_i$, with $d_i \sim C'[q_i]$. By completeness, it must be the case that there exist $\varphi_i(Z), D'_i[Z], q'_i$ with $D[X] \xrightarrow{\varphi_i(Z)}_a D'_i[Z]$ such that $p_i \models \varphi_i(q'_i)$ and $D'_i[q'_i] = d_i$, meaning that in general, according to $\sim_{\text{univ}}$, 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 $D[X] \xrightarrow{\varphi_i(Z)}_a D'_i[Z]$ is dependent on the components p_i and q_i that $C[X]$ is going to use. Thus, the difference between the symbolic and the universal approach is essentially 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 $D[X] \xrightarrow{\varphi_i(Z)}_a D'_i[Z]$ in $\sim_{\text{univ}}$, after in $\sim_{\text{strict}}$).

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.

The discussion about semantic redundancy also suggests that syntactical formats are not enough for guaranteeing that exact approximations of $\sim_{\text{univ}}$ can be inferred. Indeed, the next example shows that even De Simone format cannot ensure that $\sim_{\text{strict}} = \sim_{\text{univ}}$.

Example 3. Let us extend finite CCS with the operators $\mathit{one}_a(\_)$, $\mathit{stop}(\_)$, and with the SOS rule

$$\frac{P \rightarrow_\mu Q}{\mathit{one}_a(P) \rightarrow_a \mathit{stop}(Q)}$$

The resulting calculus CCS* adheres to the De Simone format. One can easily verify that the processes $C[X] = a.0 + a.b.0 + a.\mathit{one}_b(X)$ and $D[X] = a.0 + a.b.0 +$

stop(X) are universally bisimilar, but in the STS generated by $\mathit{Prog}_{\mathrm{A}}(\mathrm{CCS}^*)$ (see § 2), they are not strict bisimilar (intuitively, because instantiation is dynamic in the symbolic bisimulation game, while it is decided once and forever for $\sim_{\mathrm{univ}}$).

5 Symbolic Trace Semantics

Bisimilarity relates states that have "the same" branching structure. Often this feature is not directly relevant to the abstract view of the system, provided that the states can perform "the same" sequences of transitions. To this purpose, *trace semantics*—following the terminology introduced in [23]—are sometimes preferred to bisimilarity. In this section we define a hierarchy of symbolic trace semantics.

A variety of different (decorated) trace semantics has been studied in the literature (e.g., ready traces, failure traces, completed traces, accepting traces, see [2] for an overview), each relying on particular interleaved sequences of actions and state predicates. Here we just consider the basic case of finite traces (hereafter simply called traces), where finite sequences of actions are observed.

Given a component $p \in \mathcal{P}$, a *trace* of p is a finite sequence $\varsigma = a_1 a_2 \cdots a_n \in \Lambda^*$ such that there exist n components $p_1, \ldots, p_n$ with $p{\rightarrow}_{a_1}p_1{\rightarrow}_{a_2}\cdots{\rightarrow}_{a_n}p_n$ (abbreviated $p{\rightarrow}_{\varsigma}p_n$ or just $p{\rightarrow}_{\varsigma}$). The *trace language* of p is the set $\mathbf{L}(p) = \{\varsigma \in \Lambda^* \mid p{\rightarrow}_{\varsigma}\}$. Two components p and q are *trace equivalent*, written $p \simeq q$, if $\mathbf{L}(p) = \mathbf{L}(q)$. Quite obviously, for all components $p, q \in \mathcal{P}$, if $p \sim q$ then $p \simeq q$ (but the converse implication does not hold in general). As in the case of bisimilarity, the natural way of lifting trace equivalence to coordinators consists of comparing all their closed instances, defining $C[X] \simeq_{\mathrm{univ}} D[X]$ iff for all $p \in \mathcal{P}$, $C[p] \simeq D[p]$.

A different notion of trace equivalence for coordinators is readily obtained if an STS for the calculus is available. In fact, symbolic traces can be straightforwardly defined as sequences of (formula,action)-pairs.

Definition 7 ($\simeq_{\mathrm{strict}}$). *A* symbolic trace *of a coordinator* $C[X]$ *in an* STS $\mathcal{S}$ *is a finite sequence* $\zeta = \langle\varphi_1, a_1\rangle\langle\varphi_2, a_2\rangle \cdots \langle\varphi_m, a_m\rangle \in (\Phi \times \Lambda)^*$, *where* Φ *is the set of formulae in* L_{PC}, *such that there exist* m *coordinators* $C_1[X_1], \ldots, C_m[X_m]$ *with* $C[X] \xrightarrow{\{\varphi_1\}}_{a_1} C_1[X_1] \xrightarrow{\{\varphi_2\}}_{a_2} \cdots \xrightarrow{\{\varphi_m\}}_{a_m} C_m[X_m]$, *(abbreviated* $C[X]{\rightarrow}_{\zeta}C_m[X_m]$ *or just* $C[X]{\rightarrow}_{\zeta}$*). The* strict trace language *of* $C[X]$ *is the set* $\mathbf{L}(C[X]) = \{\zeta \in (\Phi \times \Lambda)^* \mid C[X]{\rightarrow}_{\zeta}\}$. *Two coordinators* $C[X]$ *and* $D[X]$ *are* strict trace equivalent, *written* $C[X] \simeq_{\mathrm{strict}} D[X]$, *if* $\mathbf{L}(C[X]) = \mathbf{L}(D[X])$.

We have $\sim_{\mathrm{strict}} \Rightarrow \simeq_{\mathrm{strict}}$ (see Theorem 2 at the end of the current section), and the inclusion is proper, as shown by the next example.

Example 4. Consider the calculus SCM, a restriction-free version of the ambient calculus [11] with asynchronous CCS-like communication, whose set of processes $\mathcal{P}$ is defined in Fig. 4. The parallel operator $|$ is associative and commutative, with 0 the identity, $a, b, \ldots$ are channels and $n, m, \ldots$ ambient names. The operational semantics of SCM, defined by SOS rules, states that, in the ambient calculus

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

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

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

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

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

Fig. 4. Syntax, associated logic and operational semantics of SCM

style, processes can move in, move out and open environments, and also asynchronously communicate within them. Being $\Lambda = \{\tau\}$, transition labels are not relevant and are omitted, i.e., we write $\rightarrow$ and $\text{-}\{\varphi\}\!\!\rightarrowtail$ in place of $\rightarrow_\tau$ and $\text{-}\{\varphi\}\!\!\rightarrowtail_\tau$. Figure 4 also shows the formulae φ of the associated logic $\mathsf{L}_{\mathrm{SCM}}$, where X is a process variable, n an ambient name and $\alpha \in \{a, \bar{a}, \mathit{open}\ n, \mathit{in}\ n, \mathit{out}\ n\}$. Since transitions are unlabelled, the modal operator does not mention any action.

Let us consider $C[X] = m[a.(a.0|b.0)|X]$ and $D[X] = m[a.0|a.b.0|X]$, and the STS generated by $\mathit{Prog}_{\mathrm{A}}(\mathrm{SCM})$ (see § 2). It holds $C[X] \not\sim_{\mathrm{strict}} D[X]$, since, for instance, the transition $C[X] \text{-}\{\bar{a}|Y\}\!\!\rightarrowtail C'[Y] = m[a.0|b.0|Y]$ could only be simulated by the transition $D[X] \text{-}\{\bar{a}|Y\}\!\!\rightarrowtail D'[Y] = m[a.b.0|Y]$, but $C'[Y] \not\sim_{\mathrm{strict}} D'[Y]$ (since $D'[X]$ can not simulate $C'[X] \text{-}\{\bar{b}|Z\}\!\!\rightarrowtail m[a.0|Z]$). On the other hand, $C[X] \simeq_{\mathrm{strict}} D[X]$. In fact, either $C[X] \text{-}\{\bar{a}|Y_1\}\!\!\rightarrowtail C_1[Y_1] \text{-}\{\bar{a}|Y_2\}\!\!\rightarrowtail C_2[Y_2] \text{-}\{\bar{b}|Y_3\}\!\!\rightarrowtail m[Y_3]$ or $C[X] \text{-}\{\bar{a}|Y_1\}\!\!\rightarrowtail C_1[Y_1] \text{-}\{\bar{b}|Y_2\}\!\!\rightarrowtail C_3[Y_2] \text{-}\{\bar{a}|Y_3\}\!\!\rightarrowtail m[Y_3]$, for obvious $C_1[Y_1], C_2[Y_2]$ and $C_3[Y_2]$, and hence, missing the label components, the language $\mathbf{L}(C[X])$ is

$$\{\lambda\ ,\ \bar{a}|Y_1\ ,\ \bar{a}|Y_1\ \bar{a}|Y_2\ ,\ \bar{a}|Y_1\ \bar{b}|Y_2\ \} \cup \{\bar{a}|Y_1\ \bar{a}|Y_2\ \bar{b}|Y_3\ ,\ \bar{a}|Y_1\ \bar{b}|Y_2\ \bar{a}|Y_3\} \cdot \mathbf{L}(m[Y_3]),$$

where "$\cdot$" is language concatenation and λ is the empty trace. The language $\mathbf{L}(D[X])$ is the same as $\mathbf{L}(C[X])$.

An alternative notion of symbolic trace can be introduced by noting that formulae φ and ψ labelling two consecutive transitions can be composed by replacing the variable occurring in φ with the formula ψ.

Definition 8 (Tight Traces). *Let $\zeta = \langle\varphi_1, a_1\rangle\langle\varphi_2, a_2\rangle \cdots \langle\varphi_m, a_m\rangle \in (\Phi \times \Lambda)^*$ be a symbolic trace of a coordinator $C[X]$. The corresponding* tight trace *is the pair $\mathit{comp}(\zeta) = (\varphi, a_1a_2\cdots a_m) \in \Phi \times \Lambda^*$, where $\varphi = \varphi_1[\varphi_2[\ldots[\varphi_m/X_{m-1}]\ldots/X_2]/X_1]$.*

Tight traces can now be used to better approximate $\simeq_{\mathrm{univ}}$.

Definition 9 ($\simeq_{\mathrm{stight}}$). *The* strict tight trace language *of $C[X]$ is $\mathbf{C}(C[X]) = \{\rho \in \Phi\times\Lambda^* \mid \exists\zeta \in \mathbf{L}(C[X]).\ \rho = \mathit{comp}(\zeta)\}$. Two coordinators $C[X]$ and $D[X]$ are* strict tight trace equivalent*, written $C[X] \simeq_{\mathrm{stight}} D[X]$, if $\mathbf{C}(C[X]) = \mathbf{C}(D[X])$.*

Since $comp(.)$ is a function, $\simeq_{\text{strict}} \Rightarrow \simeq_{\text{stight}}$ (Theorem 2). As shown by the next example the inclusion is proper (since different symbolic traces can give rise to the same tight trace).

Example 5. Consider the calculus FOO, defined over the unsorted signature $\Sigma = \{a, f(.), g(.), h(.), l(.), k(.)\}$, with the rules:

$$\begin{array}{cc} f(h(X)) \rightarrow_a h(X) & g(l(X)) \rightarrow_a l(X) \\ h(X) \rightarrow_b X & l(h(X)) \rightarrow_b X \\ f(X) \rightarrow_a k(X) & g(l(h(X))) \rightarrow_a k(X) \end{array}$$

From the symbolic transitions below, generated by $Prog_{\mathrm{A}}(\mathrm{FOO})$, it is easy to see that $f(X) \not\simeq_{\text{strict}} g(l(X))$, while $f(X) \simeq_{\text{stight}} g(l(X))$ (the traces $\langle h(Y), a\rangle\langle Z, b\rangle$ and $\langle Y, a\rangle\langle h(Z), b\rangle$ collapse in the tight trace $\langle h(Z), a\ b\rangle$).

$$\begin{array}{lllll} f(X) & \xrightarrow{\{h(Y)\}}_a & h(Y) & \xrightarrow{\{Z\}}_b & Z \ldots \\ g(l(X)) & \xrightarrow{\{Y\}}_a & l(Y) & \xrightarrow{\{h(Z)\}}_b & Z \ldots \end{array} \qquad \begin{array}{l} f(X) \xrightarrow{\{Y\}}_a k(Y) \\ g(l(X)) \xrightarrow{\{h(Y)\}}_a k(Y) \end{array}$$

As it happens for bisimilarity, the requirement of exact match between the formulae observed in a trace can be relaxed for spatial formulae.

Definition 10 (Saturated Trace). *A* saturated trace *of $C_1[X_1]$ is a finite sequence $\zeta = \langle\varphi_1, a_1\rangle\langle\varphi_2, a_2\rangle \cdots \langle\varphi_m, a_m\rangle \in (\Phi \times \Lambda)^*$, such that there exist m coordinators $C_2[X_2], ..., C_{m+1}[X_{m+1}]$ and m spatial formulae $\psi_1, ..., \psi_m$ with:*

1. $C_i[X_i] \xrightarrow{\{\varphi'_i\}}_{a_i} C'_i[Y_i]$,
2. $C_i[X_i] = C'_{i-1}[\psi_{i-1}]$,
3. $\varphi_i = \varphi'_i[\psi_i/Y_i]$,

for all $i \in [1, m]$. The saturated trace language *of $C[X]$ is the set $\mathbf{S}(C[X])$ of its saturated traces.*

A saturated trace in $\mathcal{S}$ is basically a symbolic trace in the STS obtained from $\mathcal{S}$ by adding for each symbolic transition $C[X] \xrightarrow{\{\varphi\}}_a C'[Y]$ all of its instances, i.e., a transition $C[X] \xrightarrow{\{\varphi[\psi/Y]\}}_a C'[\psi]$ for any spatial formula ψ.

Definition 11 ($\dot{\simeq}_{\text{large}}$). *$C[X]$ and $D[X]$ are* large trace pre-equivalent*, written $C[X] \dot{\simeq}_{\text{large}} D[X]$ if $\mathbf{L}(C[X]) \subseteq \mathbf{S}(D[X])$ and $\mathbf{L}(D[X]) \subseteq \mathbf{S}(C[X])$.* Large trace equivalence *$\simeq_{\text{large}}$ is the transitive closure of $\dot{\simeq}_{\text{large}}$.*

Analogously to large bisimilarity, large trace pre-equivalence $\dot{\simeq}_{\text{large}}$ might not be transitive. Hence its transitive closure is considered to obtain an equivalence.

Finally, to overcome syntactic redundancy, a notion of symbolic trace equivalence can be defined exploiting irredundant transition (see Definition 5).

Definition 12 ($\dot{\simeq}_{\text{irred}}$). *Let $\mathbf{I}(C[X])$ be the subset of $\mathbf{L}(C[X])$ containing traces composed by irredundant transitions only. Two coordinators $C[X]$ and $D[X]$ are* irredundant trace pre-equivalent*, written $C[X] \dot{\simeq}_{\text{irred}} D[X]$ if $\mathbf{I}(C[X]) \subseteq \mathbf{L}(D[X])$ and $\mathbf{I}(D[X]) \subseteq \mathbf{L}(C[X])$.* Irredundant trace equivalence *$\simeq_{\text{irred}}$ is the transitive closure of $\dot{\simeq}_{\text{irred}}$.*

The "tight" versions of $\dot{\simeq}_{\text{large}}$ and $\dot{\simeq}_{\text{irred}}$ can be defined as in the case of the strict equivalence, by resorting to the corresponding tight trace languages (see Definition 8). This leads to the trace pre-equivalences $\dot{\simeq}_{\text{ltight}}$ and $\dot{\simeq}_{\text{itight}}$ refined by the original ones, i.e., such that $\dot{\simeq}_{\text{large}} \subseteq \dot{\simeq}_{\text{ltight}}$ and $\dot{\simeq}_{\text{irred}} \subseteq \dot{\simeq}_{\text{itight}}$.

The theorem below clarifies all the relationships between all bisimilarities and trace semantics introduced so far. As expected, each kind of bisimilarity is finer than the corresponding trace semantics. The diamond involving the strict, large, irredundant and universal relations also holds for trace semantics.

Theorem 2. *Given any sound and complete* STS *for a process calculus PC, the relationships indicated in Fig. 1(b) hold, where arrows represent subset inclusion.*

6 Concluding Remarks

We introduced in [5] a methodology for reasoning about the operational and abstract semantics of open systems, viewed as coordinators in suitable process calculi, with focus on bisimilarity. Here, we have analysed how redundancy in STS may influence the quality of the approximation of universal bisimilarity $\sim_{\text{univ}}$, and we have provided a hierarchy of symbolic bisimilarities (Fig. 1(a)), where alternative equivalences for approximating $\sim_{\text{univ}}$ are proposed and studied. The approach has been extended to non-branching semantics, and, correspondingly, a hierarchy of symbolic equivalences (Fig. 1(b)) has been established.

As a matter of future investigation, we plan to develop the treatment of names and name restriction in order to deal with open systems where fresh or secret resources are a main concern. In particular, the notion of STS and the underlying process logic should be extended to deal with a logical notion of freshness, possibly taking inspiration from [9, 10]. The higher-order unification mechanism of λ-Prolog [28] could provide a convenient framework for the construction of the STS.

In the setting of name-based calculi, the openness of a system can involve not only process variables, but also communication on shared channels. This view is not in conflict with our approach, but it rather suggests an appealing direction for the future development of our work. Also for value-passing and name-based calculi, transition labels are typically structured. As shown in [22], this fact can be profitably used in a symbolic approach to define tractable behavioural equivalences, and, although labels can always be seen as a plain set, we plan to extend our symbolic approach to cope with structured transition labels.

Symbolic equivalences are intended as means for providing tractable approximations of corresponding equivalences defined by universal quantification over the set of components. Hence, on the applicative side, we expect some outcomes in the direction of the automated verification of open systems. Specifically, we are developing software tools, which, exploiting the Prolog program associated to an SOS specification, support the automated verification of symbolic equivalences. A first prototype tool, called SEA (Symbolic Equivalence Analyzer), has been developed in [30].

Pursuing further the Prolog-based algorithmic construction of STSs, we plan to investigate the use of *meta* Logic Programming for the programmable definition of transitions, and thus more specific (automated) reasoning over the structure of a PC. Moreover, also *abductive* Logic Programming is worth being considered for 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?", which is typically relevant in open and dynamic system engineering [4, 3].

Related Work. The notion of STS has been influenced by several related formalisms. Symbolic approaches to behavioural equivalences can be found in [22, 34], while the idea of using spatial logic formulae as an elegant mathematical tool for combining structural and behavioural constraints has been separately proposed in [12, 18]. Many different kinds of labelled transition systems for coordinators have been previously proposed in the literature (e.g., structured transition systems [15], context systems [25], tile logic [20], conditional transition systems [32]). Roughly, the distinguishing feature of our approach is the greater generality of symbolic transition labels which account for spatial constraints over unspecified components.

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. For a CCS fragment, early studies about terms with variables [21, 27] have shown that the presence of symbolic actions can be helpful in proving the completeness of axioms for bisimilarity. This work could be inspiring for addressing analogous issues in other calculi.

Some close relations exist also with the work on *modal transition systems* [24], 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 [14], which mixes CCS and a form of μ-calculus, to allow the logical specification of some components of the system. 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 describe their semantics by using the logic to characterise synthetically their possible transitions.

Process calculi have been traditionally used for cryptographic protocol verification, exploiting symbolic semantics for dealing with the infiniteness of the attacker models (typically due to the unconstrained generative power of intruders), see e.g. [1, 13] and especially the unification-based approach [7]. Such similarities suggest possible applications of our framework to security-oriented calculi.

The problem of the universal quantification over components in the definition of behavioural equivalences for open systems has its dual counterpart in the contextual closure needed when the bisimilarity on components $\sim$ is not a congruence and one defines the largest congruence $\simeq$ contained in $\sim$, by letting $p \simeq q$ if for all contexts $C[.]$, $C[p] \sim C[q]$. To avoid universal quantification on contexts, several authors (see [36, 26, 35]) propose a symbolic transition system for components whose labels are the "minimal" contexts needed by the component in order to evolve. Understanding to which extent this duality can be pursued and exploited is an interesting direction for future research.

References

1. M. Abadi and M.P. Fiore. Computing symbolic models for verifying cryptographic protocols. *Proc. 14th IEEE Computer Security Foundations Workshop*, pp. 160–173. IEEE Computer Society Press, 2001.
2. L. Aceto, W.J. Fokkink, and C. Verhoef. Structural operational semantics. *Handbook of Process Algebra*, pp. 197–292. Elsevier Science, 2001.
3. R. Allen and D. Garlan. A formal basis for architectural connectors. *ACM Transactions on Software Engineering and Methodology*, 3(6):213–249, 1997.
4. L.F. Andrade, J.L. Fiadeiro, L. Gouveia, G. Koutsoukos, and M Wermelinger. Coordination for orchestration. *Proc. COORDINATION 2002*, *LNCS* 2315, pp. 5–13. Springer, 2002.
5. P. Baldan, A. Bracciali, and R. Bruni. Bisimulation by unification. *Proc. AMAST 2002*, *LNCS* 2422, pp. 254–270, Springer 2002.
6. P. Baldan, A. Bracciali, and R. Bruni. Symbolic equivalences for open systems. Technical Report TR-03-16, Department of Computer Science, University of Pisa, 2003.
7. M. Boreale. Symbolic trace analysis of cryptographic protocols. *Proc. ICALP'01*, *LNCS* 2076, pp. 667–681. Springer, 2001.
8. 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.
9. L. Caires and L. Cardelli. A spatial logic for concurrency (part I). In *Proc. TACS 2001*, *LNCS* 2215, pp. 1–37. Springer, 2001.
10. L. Caires and L. Cardelli. A spatial logic for concurrency (part II). In *Proc. CONCUR 2002*, *LNCS* 2421, pp. 209–225. Springer, 2002.
11. L. Cardelli and A.D. Gordon. Mobile ambients. *Proc. FoSSaCS'98*, *LNCS* 1378, pp. 140–155. Springer, 1998.
12. L. Cardelli and A.D. Gordon. Anytime, anywhere. modal logics for mobile ambients. In *Proc. POPL 2000*, pp. 365–377. ACM, 2000.
13. E.M. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. PROCOMET'98*, Chapmann & Hall, 1998.
14. R. Cleaveland and G. Lüttgen. A logical process calculus. In *ENTCS*, 2002.
15. A. Corradini and U. Montanari. An algebraic semantics for structured transition systems and its application to logic programs. *Theoret. Comput. Sci.*, 103:51–106, 1992.
16. R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: a kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.
17. R. De Simone. Higher level synchronizing devices in MEIJE–SCCS. *Theoret. Comput. Sci.*, 37:245–267, 1985.

18. J.L. Fiadeiro, T. Maibaum, N. Martí-Oliet, J. Meseguer, and I. Pita. Towards a verification logic for rewriting logic. *Proc. WADT'99*, *LNCS* 1827, pp. 438–458. Springer, 2000.
19. R. Focardi and R. Gorrieri Classification of Security Properties (Part I: Information Flow) FOSAD'01 - Tutorial Lectures, *LNCS* 2171, pp. 331–396. Springer, 2001.
20. F. Gadducci and U. Montanari. The tile model. *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pp. 133–166. MIT Press, 2000.
21. R. van Glabbeek. A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In *Proc. MFCS'93*, *LNCS* 711, pp 473-484, Springer, 1993.
22. M. Hennessy and H. Lin. Symbolic bisimulations. *Theoret. Comput. Sci.*, 138:353–389, 1995.
23. C.A.R. Hoare. A model for communicating sequential processes. *On the Construction of Programs*. Cambridge University Press, 1980.
24. K. G. Larsen and B. Thomsen. A modal process logic. In *Proceedings of LICS*, pages 203–210. IEEE, 1988.
25. K.G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. *Proc. ICALP'90*, *LNCS* 443, pp. 526–539. Springer, 1990.
26. J.J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. *Proc. CONCUR 2000*, *LNCS 1877*, pp. 243–258. Springer, 2000.
27. R. Milner. A complete axiomatisation for observational congruence of finite-state behaviours. *Information and Computation*, 81:227–247, 1989.
28. D. Miller and G. Nadathur. Higher-order logic programming. *Handbook of Logics for Artificial Intelligence and Logic Programming*, volume 5, pp. 499–590. Clarendon Press, 1998.
29. R. Milner, J. Parrow, and J. Walker. A calculus of mobile processes, I and II. *Inform. and Comput.*, 100(1):1–40,41–77, 1992.
30. R. Nunziato. Sviluppo dell'applicazione SEA per la verifica di sistemi aperti. Master Thesis, Department of Computer Science, University of Pisa, 2003. (In Italian.)
31. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Computer Science Department, 1981.
32. A. Rensink. Bisimilarity of open terms. *Inform. and Comput.*, 156(1-2):345–385, 2000.
33. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, LFCS, University of Edinburgh, 1993. CST-99-93 (also published as ECS-LFCS-93-266).
34. D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Inform.*, 33:69–97, 1996.
35. V. Sassone and P. Sobocinski. Deriving bisimulation congruences using 2-categories. In *Nordic Journal of Computing*, volume 10. Elsevier, 2002.
36. P. Sewell. From rewrite rules to bisimulation congruences. *Proc. CONCUR'98*, *LNCS* 1466, pp. 269–284. Springer, 1998.