A Symbolic Semantics for the π-calculus - Extended Abstract -*

Michele Boreale and Rocco De Nicola

Dipartimento di Scienze dell'Informazione
Università di Roma "La Sapienza"
Email: michele@dsi.uniroma1.it, denicola@vm.cnuce.cnr.it

Symbolic transition systems are used as a basis for giving a new semantics of the π-calculus. This semantics is more amenable to automatic manipulation and sheds new light on the logical differences among different forms of bisimulation over dynamic process algebras. Symbolic transitions have the form $P \xrightarrow{\phi, \alpha} P'$, where, intuitively, ϕ is a *boolean constraint* over names that has to hold for the transition to take place, and α is a π-calculus action; e.g., $[x = y]\alpha.P \xrightarrow{[x=y], \alpha} P$ says that action α can be performed under any interpretation of names satisfying $x = y$. A symbolic bisimulation is defined on top of the symbolic transition system and it is shown that it captures the standard ones. Finally, a complete proof system is defined for symbolic bisimulation.

1 Introduction

The π-calculus [MPW92] is a widely studied process description language with primitives for expressing the exchange of channels names (or simply *names*) among processes. The exchanged names can also be tested for identity and this can be exploited to control instructions flow. These features permit a natural description of systems with dynamic linking.

Like traditional static process algebras, the π-calculus has undergone severe scrutiny and semantics for it have been proposed, relying on the standard notions of bisimulation [MPW92, PS93] and testing [Hen91, BD92]. However, theoretical studies to support equivalence checking have just begun; in [San93], with efficiency motivations, a new form of π-calculus bisimulation, called *open*, is proposed and studied.

In this paper, building on previous work by Hennessy and Lin on static value-passing process algebras [HL92, HL93], we provide a framework that yields alternative "efficient" characterization for various π-calculus bisimulation-based equivalences and a complete proof system to reason about them. An additional advantage of this framework is that it sheds light on the conceptual differences among known different forms of π-calculus bisimulation. Our attention will be confined to strong bisimulations, but we do not see any serious obstacle in extending our results to the weak ones.

* To appear in *Prc. Concur '94.*

The basic theory of bisimulation for the π-calculus has been introduced in [MPW92]. The fundamental notion is that of *ground* bisimulation; it has the same conceptual simplicity of the CCS one and suggests a natural strategy for equivalence checking. However, due to name passing, the definition-based verification technique runs into serious efficiency problems. On input actions, a case analysis on the received names is needed to check that receiving equal names leads to "equivalent" states. To see this, consider the processes $P = a(y).P'$ and $Q = a(y).Q'$; here the input prefix operator $a(y).R$ is used to describe receipt of a name at channel a and its substitution for the formal parameter y within R. To check that P and Q are ground bisimilar, following the definition we would have to verify that P' and Q' are bisimilar for all possible instantiations of y with a name occurring free in P and Q or with a fresh name. Of course, performing multiple checks for each input action may lead to exponential explosion.

Input prefix also introduces another problem: it does not preserve ground bisimulation equivalence. This leads to considering the maximal induced *congruence* as the closure under name-substitutions of the ground equivalence [MPW92]. Checking for congruence of two terms by relying on the original definition would consist in performing several (one for each "relevant" substitution) ground equivalence checks.

A simple example is sufficient to appreciate that, when verifying π-calculus bisimulation with such straightforward techniques, many performed checks are indeed useless.

Consider the two processes

$$P_1 = a(y).P \text{ and } P_2 = a(y).([y = z]P + [y \neq z]P)$$

where y and z are distinct and the $+$ operator stands for external choice. Here, both *match* $[z = y]$, and *mismatch* operators $[z \neq y]$, are used; $[z = y]P$ stands for a process that behaves like P if z and y are syntactically equal and is blocked otherwise, while $[z \neq y]P$ has exactly the opposite meaning.

It should be immediate to establish that P_1 and P_2 are equivalent. But, checking their equivalence by directly relying on the actual definition, requires checking that P and $[y = z]P + [y \neq z]P$ are bisimilar when y is replaced by z or by each of the free names of P. This could be very costly; it is however evident that, beside y, z is the only name that matters.

A significant gain in efficiency could be obtained by finding a systematic way to prune non-essential cases when performing the case analysis. To this aim, the idea pursued in this paper is that of setting up a framework where case analysis can be performed *symbolically*. First, a new transition system for the π-calculus is introduced, where the logical constraints that make a transition possible are made explicit. Then, a new bisimulation, which we refer to as *symbolic*, is defined, that performs the case-analysis directly on these logical constraints. It will be proved that symbolic bisimulation can be used to establish the standard ones.

The symbolic transitions are of the form $P \stackrel{\phi, \alpha}{\longrightarrow} P'$, where ϕ is a *boolean constraint* over names that has to hold for the transition to take place, and α is a standard π-calculus action. A typical symbolic transition is $[x = y]\alpha.P \stackrel{[x=y],\ \alpha}{\longrightarrow} P$,

saying that action α can be performed under any interpretation of names satisfying $x = y$. A boolean constraint is in general built from the basic constraints $[x = y]$ via the standard boolean connectives.

Symbolic bisimulation is defined on the top of the symbolic transition system, in a way that is parametric over the language of boolean constraints. This leads to distinct symbolic equivalences $\simeq^{\phi}$, depending on the boolean constraint ϕ. Informally, to verify $P \simeq^{\phi} Q$, it is required to find, for each symbolic move $P \stackrel{\psi,\alpha}{\longrightarrow} P'$ of P, a case-partition of the condition $\phi \wedge \psi$, such that each subcase entails a matching symbolic move for Q. As an example, the equivalence $P_1 \simeq^{true} P_2$ of the above mentioned processes is readily verified by partitioning condition *true* as $\{[z = y], [z \neq y]\}$; this is sufficient because each of these two conditions entails that $[y = z]P + [y \neq z]P$ is equal to P. Actually, as we shall explain, the appropriate case analysis can be determined automatically.

Symbolic bisimulation is related to the standard ones by the following completeness theorem:

$P \simeq^{\phi} Q$ if and only if

for each name-substitution σ satisfying ϕ, $P\sigma$ is ground equivalent to $Q\sigma$

where a name-substitution σ *satisfies* ϕ if the result of applying σ to ϕ is a tautology; here, $P\sigma$ denotes the result of applying σ to P .

The above statement tells us that each symbolic equivalence $\simeq^{\phi}$ is the closure of the ground equivalence under all name-substitutions satisfying ϕ. Thus, for example, the congruence (i.e. the closure w.r.t. *all* substitutions) will be recovered as the symbolic bisimulation $\simeq^{true}$. Ground bisimulation equivalence of two specific processes P and Q will be instead recovered as a symbolic bisimulation $P \simeq^{\phi(P,Q)} Q$, where $\phi(P, Q)$ is a constraint imposing that all free names in P and Q be distinct.

In the paper, we also present a proof system to reason about symbolic bisimulation. The statements derivable within the system are of the form $\phi \rhd P = Q$ and the system is sound and complete in the sense that $\phi \rhd P = Q$ is derivable if and only if $P \simeq^{\phi} Q$. By taking advantage of the symbolic transitional semantics, the proof of completeness is, by and large, a symbolic version of the classical completeness proof for strong bisimulation over CCS [Mil89]. Additional complications are however introduced by the fact that the boolean condition ϕ may also constraint the communication capabilities of processes.

The symbolic characterization of the standard equivalences has also an additional advantage; it sheds new light on the conceptual difference between different forms of bisimulations for the π-calculus. Actually, in [MPW92], two forms of ground bisimulations (each inducing a different congruence) were introduced, the *early* form and the *late* one. Intuitively, they correspond to two different instantiation strategies for the formal parameter of input actions: in the first strategy (early), the instantiation is performed at the moment of inferring the input action, while in the other (late) it is performed later, when a communication is actually inferred. In open bisimulation [San93], the instantiation is delayed as much as possible; this yields an equivalence much stronger than the early and late ones. Our symbolic formulation indicates that each of the men-

tioned strategies corresponds to a different degree of generality in performing case-analysis.

Besides [San93], our work mainly relates to three papers: [HL92], [HL93] and [PS93]. In [HL92], the notion of symbolic bisimulation is introduced within a syntax-free framework, where *symbolic transition graphs* are considered; a polynomial-time verification algorithm is given and, for a version of CCS with value-passing, a completeness result similar to ours is proved. For the same value-passing language, a sound and complete proof system is presented in [HL93]. In [PS93], for the same name-passing language considered here, late and early ground bisimulation and the induced congruences are equipped with four distinct algebraic proof systems; efficiency considerations are absent.

The present paper may be viewed as the extension of [HL92] and [HL93] to a name-passing calculus, for which "efficient" characterizations of different bisimulation-based equivalences are obtained. A more detailed comparison with these and other papers is deferred to Section 5. Here we want only to stress the main reason that makes this extension non-trivial. The blurring of values and channel names, a distinctive feature of the π-calculus, allows names to appear both in the actions, in the processes and in the constraints; this gives rise to a subtle interplay between name-scoping and boolean constraining. This interplay is best revealed in the symbolic SOS rules for one of the name-binders of the π-calculus, the *restriction* operator (y). In $(y)P$, the name y is declared to be *new*, i.e. different from any other name; therefore, when we have $P \xrightarrow{\phi, \alpha} P'$ as a premise of an inference rule for $(y)P$, in the conclusion we have to discard from ϕ every assumption requiring y to be equal to other names, thus obtaining a new constraint $(y)\phi$, not containing y.

Throughout the paper, the early case will be treated in full detail, while the necessary changes of definitions and arguments for the simpler late case will be indicated time by time. The rest of the paper is organized as follows. In Section 2, after introducing the π-calculus and the standard notions of bisimulation equivalences, the symbolic transitional semantics and symbolic bisimulation are presented. Correctness and completeness of the latter w.r.t. standard bisimulations are also discussed. Section 4 presents the proof system and the corresponding theorems of soundness and completeness. Section 5 contains conclusions, comparisons with related work and suggestions for future research. Due to lack of space, proofs are just sketched; detailed proofs can be found in the full version of the paper [BD94].

2 Symbolic Semantics

In this section the π-calculus [MPW92] and its standard bisimulations will be briefly reviewed; then the new symbolic semantics will be introduced.

2.1 The π-calculus and its standard bisimulation semantics

Definition 1. *(Syntax)* Let $\mathcal{N}$ be a countable set and x, y range over it, let ϕ range over the language BF of *Boolean Formulae*:

$\phi ::= true \mid [x = y] \mid \neg\phi \mid \phi \wedge \phi$

and let α range over *actions*:

$\alpha ::= \tau$ *(silent move)* $\mid x(y)$ *(input)* $\mid \overline{x}y$*(free output)*

Let X range over a countable set of *process variables* and consider the language of *agent terms* built by means of *agent variables, inaction, action prefix, summation, boolean guard, restriction, parallel composition* and *recursion* in the following way:

$P ::= X \mid \mathbf{0} \mid \alpha.P \mid P + P \mid \phi P \mid (y)P \mid P|P \mid recX.P$

A *process* is an agent term where each occurrence of any agent variable X lies within the scope a $recX.$ operator. We let π denote the set of processes.

We fix now some basic notations. We shall use $false$ as an abbreviation for $\neg true$, $[x \neq y]$ as an abbreviation for $\neg[x = y]$ and $\phi_1 \vee \phi_2$ as an abbreviation for $\neg(\neg\phi_1 \wedge \neg\phi_2)$. *Evaluation* of a boolean formula into the set $\{true, false\}$ is defined in the expected way, once we set that for any two distinct names x and y, $[x = x]$ evaluates to $true$ and $[x = y]$ evaluates to $false$. We will write $\phi = true$ ($\phi = false$) if ϕ evaluates to $true$ ($false$); $n(\phi)$ will denote the set of names occurring in ϕ.

We use the *bound output action* $\overline{x}(y).P$, $x \neq y$, as a shorthand for $(y)(\overline{x}y.P)$. If $\alpha = x(y)$ or $\alpha = \overline{x}y$ or $\alpha = \overline{x}(y)$, we let $subj(\alpha) = x$ and $obj(\alpha) = y$. The π-calculus has two kinds of name *binders*: input prefix $x(y).P$ and restriction $(y)P$ bind the name y in P; consequently, the notions of *free names, fn(.)*, *bound names, bn(.)* and *α-equality*, $\equiv$, over both process terms, formulae and actions, are the expected ones (we define $fn(\phi) = n(\phi)$ for a boolean formula ϕ). We let $n(.) = fn(.) \cup bn(.)$.

Substitutions, ranged over by σ, ρ, are functions from $\mathcal{N}$ to $\mathcal{N}$; for any $x \in \mathcal{N}$, $\sigma(x)$ will be written as $x\sigma$. Given a substitution σ and $V \subseteq_{fin} \mathcal{N}$, we define:

- $V\sigma = \{x\sigma \mid x \in V\}$
- $dom(\sigma) = \{x \mid x\sigma \neq x\}$
- $range(\sigma) = dom(\sigma)\sigma$
- $n(\sigma) = dom(\sigma) \cup range(\sigma)$

In the rest of the paper we confine ourselves to *finite* substitutions, i.e. those σ s.t. $n(\sigma)$ is finite. If t is either an action, a formula or a process term, $t\sigma$ denotes the result of applying the substitution σ to t, i.e. the expression obtained from t by simultaneously replacing each $x \in fn(t)$ with $x\sigma$. A set $\{x_1/y_1, \ldots, x_n/y_n\} = \{\tilde{x}/\tilde{y}\}$, with the x_i's pairwise distinct, will denote the following substitution σ: $x\sigma = y_i$ if $x = x_i$ for some $i \in \{1, \ldots, n\}$, $x\sigma = x$ otherwise. We also define $fn(\sigma) = n(\sigma)$; in this way, function $fn(\,.\,)$ is defined over both names, actions, processes, formulae and substitutions.

Unless otherwise stated, we will let $x, y, \ldots$ range over $\mathcal{N}$, $\alpha, \beta, \ldots$ over the set of actions (including derived bound output), $\phi, \psi, \ldots$ over BF, $P, Q, \ldots$ over π and $\rho, \sigma, \ldots$ over substitutions.

The standard "concrete" transitional semantics of π is given in Table 1. By following [PS93] we also include the (non-structural) `Alpha` rule, that permits

$$\texttt{Act}\ \frac{-}{\alpha.P \xrightarrow{\alpha} P}$$

$$\texttt{Sum}\ \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 + P_2 \xrightarrow{\alpha} P_1'} \qquad \texttt{Par}\ \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 \mid P_2 \xrightarrow{\alpha} P_1' \mid P_2}\ bn(\alpha) \cap fn(P_2) = \emptyset$$

$$\texttt{Com}\ \frac{P_1 \xrightarrow{\overline{x}z} P_1',\ P_2 \xrightarrow{x(y)} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'\{z/y\}} \qquad \texttt{Close}\ \frac{P_1 \xrightarrow{\overline{x}(y)} P_1',\ P_2 \xrightarrow{x(y)} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} (y)(P_1' \mid P_2')}$$

$$\texttt{Res}\ \frac{P \xrightarrow{\alpha} P'}{(y)P \xrightarrow{\alpha} (y)P'}\ y \notin n(\alpha) \qquad \texttt{Open}\ \frac{P \xrightarrow{\overline{x}y} P'}{(y)P \xrightarrow{\overline{x}(y)} P'}\ x \neq y$$

$$\texttt{Guard}\ \frac{P \xrightarrow{\alpha} P'}{\phi P \xrightarrow{\alpha} P'}\ \phi = true \qquad \texttt{Rec}\ \frac{P[recX.P/X] \xrightarrow{\alpha} P'}{recX.P \xrightarrow{\alpha} P'}$$

$$\texttt{Alpha}\ \frac{P \xrightarrow{\alpha} P'}{Q \xrightarrow{\beta} Q'}\ P \equiv Q,\ \alpha.P' \equiv \beta.Q'$$

Symmetric versions of `Sum`, `Par`, `Com` and `Close` are omitted.

Table 1. Standard SOS for π.

freely α-renaming actions and processes; this rule often avoids tedious side conditions in the proofs. With our transitional semantics, the definition of (standard) early ground bisimulation equivalence $\dot{\sim}$ and early bisimulation congruence $\sim$ can be given as follows:

Definition 2. *(Early Bisimulation)*

- A symmetric relation $\mathcal{R} \subseteq \pi \times \pi$ is a ground early bisimulation iff $(P, Q) \in \mathcal{R}$ and $P \xrightarrow{\alpha} P'$ with $bn(\alpha) \cap fn(P, Q) = \emptyset$, imply
  - if α is not an input action, then $\exists\, Q'$: $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{R}$.
  - if $\alpha = x(y)$, then $\forall z \in fn(P, Q, y)\ \exists\, Q' : Q \xrightarrow{\alpha} Q'$ and $(P'\{z/y\}, Q'\{z/y\}) \in \mathcal{R}$.
- $\dot{\sim} = \cup\{\mathcal{R} \mid \mathcal{R}$ is a ground early bisimulation $\}$.
- $P \sim Q$ iff $\forall\, \sigma.\ P\sigma \dot{\sim} Q\sigma$.

The *late* version of the above definition is obtained by replacing the input-clause (the second clause of the first item) by the stronger:

- if $\alpha = x(y)$, then $\exists Q' : Q \xrightarrow{\alpha} Q'$ and $\forall\, z \in fn(P,Q,y)\ (P'\{z/y\}, Q'\{z/y\}) \in \mathcal{R}$.

It turns out [MPW92] that late bisimulation is strictly finer than early bisimulation.

2.2 Symbolic semantics

Before introducing the symbolic semantics, we need to fix some additional notation for boolean formulae and substitutions.

Definition 3. *(Basic Definitions)*

- $\sigma \models \phi$ stands for $\phi\sigma = true$;
- $\phi \models \psi$ stands for $\forall\ \sigma$: $\sigma \models \phi$ implies $\sigma \models \psi$;
- $[\alpha = \beta]$ stands for

$[x = z]$	if for some y ($\alpha = x(y)$ and $\beta = z(y)$) or ($\alpha = \overline{x}(y)$ and $\beta = \overline{z}(y)$);
$[x = z] \wedge [y = w]$	if $\alpha = \overline{x}y$ and $\beta = \overline{z}w$;
$true$	if $\alpha = \beta = \tau$;
$false$	otherwise.

- $\bigvee D$, where $D = \{\phi_1, \ldots, \phi_n\} \subseteq_{fin} BF$, $n > 0$, is the boolean formula $\phi_1 \vee \ldots \vee \phi_n$. A similar notation will be used for $\bigwedge D$. Furthermore, we let $\bigvee \emptyset$ denote $false$ and $\bigwedge \emptyset$ denote $true$.
- The boolean formula $(y)\phi$ is defined by structural induction on ϕ as follows:

$$
\begin{aligned}
(y)true &= true \\
(y)[w_1 = w_2] &= [w_1 = w_2] && \text{if } y \notin \{w_1, w_2\} \\
(y)[y = y] &= true \\
(y)[y = w] &= (y)[w = y] = false && \text{if } y \neq w \\
(y)\neg\phi &= \neg(y)\phi \\
(y)(\phi_1 \wedge \phi_2) &= (y)\phi_1 \wedge (y)\phi_2
\end{aligned}
$$

The symbolic transitional semantics of π is presented in Table 2. We comment on the rules. Each symbolic rule is the counterpart of a concrete one. Intuitively, the boolean condition ϕ in $P \xrightarrow{\phi,\,\alpha} P'$ is a constraint over the free names of P that has to hold for the transition to take place. The rule $\phi - \texttt{Act}$ says that $\alpha.P$ can perform α unconditionally. In $\phi - \texttt{Com}$ and $\phi - \texttt{Close}$, the condition of matching channels ($[x = w]$) is moved into the boolean condition of the resulting symbolic transition; this is necessary to infer symbolic transitions such as: $(x(y).\mathbf{0}) \mid (\overline{w}\langle y\rangle.\mathbf{0}) \xrightarrow{[x=w],\,\tau} \mathbf{0} \mid \mathbf{0}$. The rule $\phi - \texttt{Res}$ reveals the interplay between name-scoping and constraining: in $(y)P$, the name y is declared to be *new*, i.e. different from any other name; thus the rule says that, given a symbolic transition $P \xrightarrow{\phi,\,\alpha} P'$ as a premise, under the assumption that y is new, we have to discard every assumption about the identity of y from ϕ in the conclusion;

$$\phi - \texttt{Act}\ \frac{-}{\alpha.P \xrightarrow{true,\alpha} P}$$

$$\phi - \texttt{Sum}\ \frac{P_1 \xrightarrow{\phi,\alpha} P_1'}{P_1 + P_2 \xrightarrow{\phi,\alpha} P_1'} \qquad \phi - \texttt{Par}\ \frac{P_1 \xrightarrow{\phi,\alpha} P_1'}{P_1 \mid P_2 \xrightarrow{\phi,\alpha} P_1' \mid P_2}\ bn(\alpha) \cap fn(P_2) = \emptyset$$

$$\phi - \texttt{Com}\ \frac{P_1 \xrightarrow{\phi_1,\overline{x}z} P_1',\ P_2 \xrightarrow{\phi_2,w(y)} P_2'}{P_1 \mid P_2 \xrightarrow{\phi_1 \wedge \phi_2 \wedge [x=w],\tau} P_1' \mid P_2'\{z/y\}} \qquad \phi - \texttt{Close}\ \frac{P_1 \xrightarrow{\phi_1,\overline{x}(y)} P_1',\ P_2 \xrightarrow{\phi_2,w(y)} P_2'}{P_1 \mid P_2 \xrightarrow{\phi_1 \wedge \phi_2 \wedge [x=w],\tau} (y)(P_1' \mid P_2')}$$

$$\phi - \texttt{Res}\ \frac{P \xrightarrow{\phi,\alpha} P'}{(y)P \xrightarrow{(y)\phi,\alpha} (y)P'}\ y \notin n(\alpha) \qquad \phi - \texttt{Open}\ \frac{P \xrightarrow{\phi,\overline{x}y} P'}{(y)P \xrightarrow{(y)\phi,\overline{x}(y)} P'}\ x \neq y$$

$$\phi - \texttt{Guard}\ \frac{P \xrightarrow{\psi,\alpha} P'}{\phi P \xrightarrow{\phi \wedge \psi,\alpha} P'} \qquad \phi - \texttt{Rec}\ \frac{P[recX.P/X] \xrightarrow{\phi,\alpha} P'}{recX.P \xrightarrow{\phi,\alpha} P'}$$

$$\phi - \texttt{Alpha}\ \frac{P \xrightarrow{\phi,\alpha} P'}{Q \xrightarrow{\phi,\beta} Q'}\ P \equiv Q,\ \alpha.P' \equiv \beta.Q'$$

Symmetric versions of $\phi - \texttt{Sum}$, $\phi - \texttt{Par}$, $\phi - \texttt{Com}$ and $\phi - \texttt{Close}$ are omitted.

Table 2. Symbolic SOS for π.

as a result, the new constraint $(y)\phi$ that does not contain y is obtained. An example of derivation is:

$$\phi - \texttt{Res}\ \frac{P \xrightarrow{[y=z] \vee [z=w],\tau} P'}{(y)P \xrightarrow{false \vee [z=w],\tau} (y)P'}$$

where z and w are different from y (note that $false \vee [z = w]$ is equivalent to $[z = w]$). A similar comment holds for $\phi - \texttt{Open}$. The other inference rules should be self-explanatory.

We are now set to introduce symbolic bisimulation for the π-calculus. For a variant of CCS with value passing, symbolic bisimulation has been introduced in [HL92]. The underlying intuition is that of establishing equivalence of, say, P and Q under a condition ϕ, by matching, under ϕ, symbolic transitions of P, $P \xrightarrow{\psi,\alpha} P'$, by *sets* of symbolic transitions of Q. More precisely, the condition $\phi \wedge \psi$, is partitioned into a set D of subcases, each of which entails a matching

transition for Q. Due to the treatment of restricted names in the π-calculus, here we have to take into account the case when the performed action is a bound output $\overline{x}(y)$. In this case, we impose, by adding an extra condition, that the name y be different from any known name in subsequent reasoning.

Definition 4. *(ϕ-decomposition)* Given ϕ and a finite set of formulae $D = \{\phi_1, \ldots, \phi_n\}$, we say that D is a *ϕ-decomposition* iff $\phi \models \bigvee D$.

Definition 5. *(Symbolic Early Bisimulation)*

- A family $\mathcal{F} = \{R_\phi \mid \phi \in BF\}$ of symmetric binary relations over π, indexed over the set BF of boolean formulae, is a *family of symbolic early bisimulations (FSEB)* iff for each ϕ, $(P,Q) \in R_\phi$ and $P \xrightarrow{\psi,\alpha} P'$, with $bn(\alpha) \cap fn(P,Q,\phi) = \emptyset$, imply:
  there exists a χ-decomposition D, such that for each $\phi' \in D$, there is a symbolic transition $Q \xrightarrow{\psi',\beta} Q'$ with $\phi' \models (\psi' \wedge [\alpha = \beta])$ and $(P',Q') \in R_{\phi'}$, where:

$$\chi = \begin{cases} \phi \wedge \psi \wedge \bigwedge_{z \in fn(P,Q,\phi)} [y \neq z] & \text{if } \alpha \text{ is a bound output action } \overline{x}(y) \\ \phi \wedge \psi & \text{otherwise.} \end{cases}$$

- $P \simeq^\phi Q$ iff there exists a FSEB $\{R_\psi \mid \psi \in BF\}$ such that $(P,Q) \in R_\phi$.

Note that, on input actions, no multiple instantiation of the formal parameter is required; instead, a single instantiation with a *fresh* name suffices (the "freshness" condition is $bn(\alpha) \cap fn(P,Q,\phi) = \emptyset$ [2]). The case analysis on the received value is now embodied in the decomposition D; by performing this decomposition in an appropriate way, the number of cases to deal with is in general much smaller than that arising with the original definition (as shown, e.g., by the example about the processes P_1, P_2 in the Introduction).

Actually, the appropriate decomposition can be determined automatically. In [HL92], an algorithm is presented to check symbolic bisimulation between two finite *standard* symbolic transition graphs. A symbolic graph is standard if the bound name of each bound action transition does not occur free in any ancestor-node of the transition. This amounts to requiring that only *new* names are used for bound actions when generating the graph. Given two such graphs G_1 and G_2, the algorithm calculates, in a time polynimial with the sizes of the graphs, the *most general boolean expression* under which the two graphs are equivalent, i.e. a boolean ϕ such that if $G_1 \simeq^\psi G_2$, then $\phi \models \psi$. Therefore, the equivalence problem for graphs is reduced to the implication problem for boolean expressions. Now, altough a π-calculus process, even if finite, has in general infinitely many α-equivalent transitions (due to the $\phi -$ `Alpha` rule), it is easy to see that only finitely many of them need to be considered when performing verificaton. More precisely, it is enough to consider a transition for each α-equivalence class of transitions. Starting from a π-calculus term, we can thus generate a graph that

[2] $fn(\psi)$ need not to be considered because we have $fn(\psi) \subseteq fn(P)$.

represents it and that, at least for finite processes, is finite. In order for the graph to be standard, we have also to take care of using fresh names for input and bound output transitions (by resorting, e.g., to a fresh name generator). By introducing minor modifications (that take into account the extra conditions due to bound output when determining the decomposition), we can then use Hennessy and Lin' algorithm to calculate the most general boolean expression of two π-calculus processes represented by finite symbolic transition graphs.

Symbolic Late Bisimulation is obtained by simply adding the condition $bn(\alpha) \cap n(\bigvee D) = \emptyset$ to the first item of the Definition 5: this amounts to imposing that no alternative of the decomposition depends on the "value" of the formal parameter $bn(\alpha)$, i.e. to forbidding case-analysis on the actual value of $bn(\alpha)$. The above discussion on automatic verification extends to the late case as well, by considering the late version of Hennessy and Lin' algorithm.

The following lemma, that relates standard and symbolic transitional semantics, is crucial to establish correctness and completeness of symbolic bisimulations.

Lemma 6. (Correspondence between Symbolic and Concrete SOS)

1. *If* $P \xrightarrow{\phi,\alpha} P'$, *with* $bn(\alpha) \cap fn(P,\sigma) = \emptyset$, *and* $\sigma \models \phi$, *then* $P\sigma \xrightarrow{\alpha\sigma} P'\sigma$.
2. *If* $P\sigma \xrightarrow{\alpha} P'$, *with* $bn(\alpha) \cap fn(P,\sigma) = \emptyset$, *then there exists a symbolic transition* $P \xrightarrow{\phi,\beta} P''$, *with* $\sigma \models \phi$, $\beta\sigma = \alpha$ *and* $P''\sigma = P'$.

PROOF: By transition induction on $P \xrightarrow{\phi,\alpha} P'$ and $P\sigma \xrightarrow{\alpha} P'$. □

In order to state both the the correctness and the completeness theorems, it is useful to have the following definition:

Definition 7. *(Closing $\dot{\sim}$ under ϕ)*For each $\phi \in BF$, let relation $\sim^\phi$ be: $P \sim^\phi Q$ iff $\forall\sigma$ such that $\sigma \models \phi$, $P\sigma \dot{\sim} Q\sigma$.

Theorem 8. (Correctness of Symbolic Bisim.) $P \simeq^\phi Q$ *implies* $P \sim^\phi Q$.

PROOF: Sketch. Consider the relation $\mathcal{R} = \{(P\sigma, Q\sigma) | \exists \phi.\, \sigma \models \phi \text{ and } P \simeq^\phi Q\}$. It is not difficult, by exploiting Lemma 6, to show that $\mathcal{R} \subseteq \dot{\sim}$. Precisely, one shows that $\mathcal{R}$ is a *Bisimulation up to Injective Substitutions*; in this proof technique, the clauses of usual bisimulation are weakened so to permit that matching transitions lead to states that are in the relation provided that some injective substitution is applied to them (in the present case, this is useful for bound output transitions). Details on the technique can be found in the full version of this paper [BD94] or in [San94]. □

3 Completeness of Symbolic Bisimulation

To prove the completeness theorem, we shall rely on the fact that only a suitable finite set of name-substitutions is "relevant" when working with fixed collection

of processes and formulae. All the remaining substitutions are, in fact, *variants* of the considered ones, i.e. they can be obtained by injective renaming. This is a distinctive property of the π-calculus, since it relies on the blurring of names and variables.

Theorem 9. (Completeness of Symbolic Bisim.) *$P \sim^{\phi} Q$ implies $P \simeq^{\phi} Q$.*

PROOF: Sketch. We show that $\{\sim^{\phi} \mid \phi \in BF\}$ is a family of symbolic bisimulations. Suppose that $P \sim^{\phi} Q$ and that $P \xrightarrow{\psi,\alpha} P'$. We sketch here only the case $\alpha = \tau$. We have to find a decomposition of $\phi \wedge \psi$, such that each subcase entails a matching symbolic transition for Q. Fix any σ s.t. $\sigma \models \phi \wedge \psi$. From Lemma 6 applied to $P \xrightarrow{\psi,\alpha} P'$, we obtain that $P\sigma \xrightarrow{\tau} P'\sigma$. Hence, by definition of $\sim^{\phi}$, there is a transition $Q\sigma \xrightarrow{\tau} Q' \dot{\sim} P'\sigma$. Again from Lemma 6, we obtain that there exists a transition $Q \xrightarrow{\phi_\sigma,\tau} Q_\sigma$, with $Q_\sigma\sigma = Q' \dot{\sim} P'\sigma$ and $\sigma \models \phi_\sigma$.

Now, let $V = fn(P, Q, \phi \wedge \psi)$. The idea is to consider a *finite* set of substitutions, $S = \{\sigma_1, \ldots, \sigma_k\}$, such that each substitution σ satisfying $\phi \wedge \psi$ is a variant over V of some σ_i. These are obtained as the set of those σ's satisfying $\phi \wedge \psi$ with $n(\sigma) \subseteq V \cup Y$, where Y is a suitably large finite set of names disjoint from V (Y represents a reserve of fresh names). Then, we decompose $\phi \wedge \psi$ into a set $D = \{\psi_1, \ldots, \psi_k\}$, one subcase for each $\sigma_i \in S$; more precisely $\psi_i = \chi(\sigma_i, V) \wedge \phi_{\sigma_i}$, where $\chi(\sigma_i, V)$ is a certain formula satisfied exactly by the the variants of σ_i over V. Now, for each $i \in \{1, \ldots, k\}$, from $P'\sigma_i \dot{\sim} Q_{\sigma_i}\sigma_i$ and from the fact that ψ_i is satisfied only by variants of σ_i, we can conclude that $P' \sim^{\psi_i} Q_{\sigma_i}$ (in fact, variant substitutions "behave the same" w.r.t. $\dot{\sim}$). Furthermore, by exploiting the fact that for each variant σ of σ_i, $\sigma_i \models \phi \wedge \psi$ implies $\sigma \models \phi \wedge \psi$, we can conclude that D is a $\phi \wedge \psi$-decomposition.

The cases when α is an input or α is bound output are more involved, because we have to take into account the possibility of case-analysis also over the formal parameter $bn(\alpha)$. □

We end the section by showing that also ground bisimulation $\dot{\sim}$ can be characterized in terms of the symbolic one.

Theorem 10. *$P \dot{\sim} Q$ iff $P \simeq^{\phi} Q$, where $\phi = \bigwedge_{x,y \in fn(P,Q),\, x,y \text{ distinct}} [x \neq y]$.*

PROOF: The theorem follows immediately from the correctness and completeness theorems for symbolic bisimulation and from the fact that $\dot{\sim}$ is closed under injective substitutions . □

4 The Proof System

Let us now consider the *finite* fragment of the calculus, i.e. the calculus without the $recX.$ operator zand discuss an equational axiomatization of symbolic bisimulation over it. It is well known that decidable axiomatizations cannot exist for the full language. The statements derivable within the system are guarded equations of the form $\phi \rhd P = Q$, to be read as "under ϕ, P equals Q". In the

sequel, we will write $\phi \rhd P = Q$ to mean that the equation $\phi \rhd P = Q$ is derivable within the proof system. Furthermore, we will abbreviate "$true \rhd P = Q$" simply as "$P = Q$".

The inference rules and the new relevant axioms of the proof system are presented in Table 3 and Table 4. The standard inference rules for reflexivity, symmetry and transitivity and the usual laws for Summation, Restriction and Alpha-conversion (see [MPW92]) have been omitted in this shortened version for lack of space. Our proof system can be viewed as the result of merging that of [HL93] and [PS93]. More precisely, all of the inference rules, but the *Res* rule, are taken from [HL93], while the axioms are taken from [PS93].

The *Cut* rule permits case analysis on ϕ: it says that if ϕ can be split into two subcases ϕ_1 and ϕ_2, and for each subcase we can prove $P = Q$, then $P = Q$ is derivable under ϕ. The *Res* rule exhibits the same kind of logical "hiding" of the bound name y as the rules $\phi - \texttt{Res}$ and $\phi - \texttt{Open}$ of the transitional semantics. The other rules and axioms should be self-explanatory; anyway, we refer the reader to [PS93, HL93] for explanations on their intuitive meaning.

Soundness is straightforward to prove by exploiting the correctness and completeness of symbolic bisimulation w.r.t. the standard one.

Theorem 11. (Soundness of the Proof System)$\phi \rhd P = Q$ *implies* $P \sim^{\phi} Q$.

The actual proof of completeness relies on a "customized" notion of head normal form. Each process term has a provably equivalent head normal form.

Definition 12. *(Head Normal Forms)* A process P is in *head normal form (HNF)* if it is of the form $\sum_{i \in I} \phi_i S_i$, where:

- $\{\phi_i | \, i \in I\}$ is a $true$-decomposition such that $\phi_i \wedge \phi_j = false$ for each $i, j \in I$ with $i \neq j$;
- each S_i is of the form $\sum_{j \in J_i} \alpha_j.P_j$.

Theorem 13. (Complet. of the Proof System) $P \simeq^{\phi} Q$ *implies* $\phi \rhd P = Q$.

PROOF: Sketch. The proof is by induction on the depth of P and Q under ϕ. If $P \simeq^{\phi} Q$, we can suppose, without loss of generality, that both P and Q are in HNF. We then split the condition ϕ into a decomposition D such that for each subcase $\psi \in D$:

1. under ψ, both $P = \sum_{i \in I} \alpha_i.P_i$ and $Q = \sum_{j \in J} \beta_j.Q_j$, that is P and Q are equal to some head normal form in the sense of [Mil89].
2. under ψ, all the free names appearing in some α_i or some β_j, can be treated as *constants*. More precisely, for any to such name x and y, if not $\phi \models [x = y]$ then $\phi \models [x \neq y]$.

By exploiting the above facts, the symbolic transitional semantics and the inductive hypothesis, one can show that P and Q are equal under ψ, i.e. $\psi \rhd P = Q$. Since this holds for each subcase $\psi \in D$, we can conclude by applying the *Cut* rule that $\phi \rhd P = Q$. □

$$(Congr)\ \frac{\phi \rhd P = Q}{\phi \rhd P' = Q'}$$ where $P' = Q'$ stands for either of $\tau.P = \tau.Q$, $\overline{x}y.P = \overline{x}y.Q$, $\psi P = \psi Q$, $P + R = Q + R$, $P|R = Q|R$.

$$(Res)\ \frac{\phi \rhd P = Q}{(y)\phi \rhd (y)P = (y)Q}$$

$$(Inp)\ \frac{\phi \rhd \sum_{i\in I} \tau.P_i = \sum_{i\in I} \tau.Q_i}{\phi \rhd \sum_{i\in I} x(y).P_i = \sum_{i\in I} x(y).Q_i}\ y \notin n(\phi)$$

$$(Guard)\ \frac{\phi \wedge \psi \rhd P = Q,\ \phi \wedge \neg\psi \rhd Q = \mathbf{0}}{\phi \rhd \psi P = Q}$$

$$(False)\ \frac{-}{false \rhd P = Q}$$

$$(Cut)\ \frac{\phi_1 \rhd P = Q,\ \phi_2 \rhd P = Q}{\phi \rhd P = Q}\qquad \phi \models \phi_1 \vee \phi_2$$

$$(Axiom)\ \frac{-}{true \rhd P = Q}$$ for each axiom $P = Q$

Table 3. Inference Rules of the Proof System

A sound and complete proof system for late bisimulation can be obtained by replacing the Inp rule of the considered system with the simpler rule:

$$(Inp - L)\frac{\phi \quad P = Q}{\phi \quad x(y).P = x(y).Q}\ y \notin n(\phi).$$

$(Subst)$ $[x = y]\alpha.P = [x = y]\alpha\{x/y\}.P$

$(Res2)$ $(y)(\phi P) = ((y)\phi)(y)P$

(Exp)
Suppose $P \equiv \sum_{i\in I} \phi_i\alpha_i.P_i$ and $Q \equiv \sum_{j\in J} \psi_i\beta_i.Q_i$.
Suppose that no α_i (resp. β_j) binds a name free in Q (resp. P).

$$P \mid Q = \sum_{i\in I} \phi_i\alpha_i.(P_i \mid Q) + \sum_{j\in J} \psi_j\beta_j.(P \mid Q_j) + \sum_{\alpha_i\, opp\, \beta_j} (\phi_i \wedge \psi_j \wedge [x_i = y_j])\tau.R_{ij}$$

where $\alpha_i\, opp\, \beta_j$ and R_{ij} are defined as follows:
1. $\alpha_i = \overline{x_i}z$ and $\beta_j = y_j(y)$; then $R_{ij} = P_i \mid Q_j\{z/y\}$
2. $\alpha_i = \overline{x_i}(y)$ and $\beta_j = y_j(y)$; then $R_{ij} = (y)(P_i \mid Q_j)$
3. The converse of 1.
4. The converse of 2.

Table 4. Relevant Axioms of the Proof System.

5 Conclusions and Related Work

A symbolic transitional semantics for the π-calculus has been introduced and, on top of it, a notion of symbolic bisimulation has been defined, amenable to efficient checking. Symbolic bisimulation has then been related to the standard bisimulations of the π-calculus. A consequence of this is that more efficient checking of early and late bisimulations is possible. A sound and complete proof system for symbolic bisimulation has then been provided.

The symbolic characterization of the bisimulations has another major benefit: it sheds new light on the logical difference between various π-calculus bisimulations, based on different instantiation strategies, such as early, late and open. It is not difficult to see that different instantiation strategies correspond to different degrees of generality in the case analysis. Indeed, early bisimulation is the most general (and natural) one, since no restriction is imposed on the case-decomposition D. Late bisimulation is obtained by adding the requirement that the formal parameter of the input action ($bn(\alpha)$) does not appear in D, i.e. by forbidding case-analysis on the actual value of the formal parameter. Open bisimulation [San93] is only defined over the language without negation (hence essentially without mismatch $[x \neq y]$). It appears, but this needs to to be worked out in full detail, that open bisimulation can be obtained from the early symbolic one by completely omitting case analysis; this amounts essentially to requiring that D be a singleton. Extending open bisimulation to a language with mismatch is an interesting open problem of [San93]. It could be solved within our symbolic framework. It appears that a meaningful "conservative" extension of open bisimulation to the richer language can be obtained by requiring a limited

form of case-analysis.

The original idea of symbolic bisimulation has been presented in [HL92]. There, a polynomial verification algorithm is proposed for a class of symbolic transition graphs and a theorem relating symbolic bisimulations to concrete bisimulations over a version of CCS with value-passing is presented. In [HL93], the same language has then been equipped with a sound and complete proof system. The results obtained by Hennessy and Lin are the direct inspiration of our work but they cannot be directly extended to the π-calculus for two main reasons:

1. the blurring of the distinction between variables, values and channels proper of the π-calculus;
2. the absence of a specific language for boolean constraints in the work by Hennessy and Lin.

It is easier to deal with a static value-passing process algebra, because channel names are neatly separated from the exchanged values and thus channels do not appear in the constraints. Of course, this is no longer true in a name-passing calculus, where a subtle interplay between name-scoping and boolean constraining is present. An example of such interplay is offered by the symbolic SOS rules for the restriction operator.

The symbolic framework of [HL92] and [HL93] is parametrised on the language of boolean constraints, in other words they do not have a specific language for boolean constraints. In order to establish the relationship between symbolic and concrete bisimulation, they just assume the existence of an extremely expressive language, capable of describing any given collection of environments (associations of variables with values). This is admittedly [HL92] a very strong requirement. It is at least not obvious, in presence of non-trivial types of values, that such a language exists. Here, we had to consider a *specific* language (BF) and had to deal with name substitutions rather than environments. Indeed, it must be said that our solution heavily depends on the specific features of the π-calculus: only finitely many substitutions are important when dealing with a fixed set of constraints and processes. This property does not hold for languages that, besides names, permit exchanging other kinds of values (e.g. integers) and make use of predicates (e.g. $\leq$) over them.

In [PS93], the ground equivalence and the corresponding congruence, for the early and late cases, are separately axiomatized, via four distinct algebraic proof systems. Confining ourselves to one specific form of bisimulation (either early or late), the main differences between our proof system and theirs can be summarized as follows. They consider the ground equivalence and the congruence separately; in our framework, all the equivalences obtainable as substitution-closure of the ground one (including, as particular cases, the ground equivalence itself and the congruence) are considered at once. As a consequence, it is possible to reason about each such equivalence, just by selecting the appropriate ϕ. Furthermore, at least when considering late ground equivalence, we gain in efficiency. If it has to be proven that $x(y).P$ is ground bisimilar to $x(y).Q$, within our framework it just suffices to derive $\phi \rhd P = Q$, for some ϕ not containing

y and not stronger than the constraint given by our characterization of $\dot{\sim}$ in terms of $\simeq^{\phi}$. Within the framework of [PS93], it is instead needed to apply the input-prefix rule:

$$IP: \frac{\forall z \in fn(P,Q,y).\ P\{^z/_y\} = Q\{^z/_y\}}{x(y).P = x(y).Q}$$

whose premise *always* requires as many sub-proofs as the cardinality of $fn(P,Q,y)$. This example shows that making reasoning assumptions explicit can often avoid a number of useless checks. An accurate comparison between the two approaches w.r.t. efficient deduction strategies would be interesting.

Finally, our work is somewhat related to [Dam93] and to [FMQ94], where different kinds of symbolic transitional semantics for the π-calculus have been presented. In [Dam93], a symbolic semantics is used as a basis for developing a model checker; first-order (rather than boolean) formulae are utilized; in the operational rules for the restriction operator, the "hiding" of a name y in a formula is modeled using an existential quantifier $\exists y$. The aim of [FMQ94] is to define a general framework, within which the different kinds of instantiation strategies (such as early, late, open) can be described just by instantiating certain parameters. The problem of efficiently representing the considered equivalences is not tackled.

References

[BD92] M. Boreale and R. De Nicola. Testing equivalence for mobile processes. Technical Report SI 92 RR 04, Dipartimento di Scienze dell'Informazione Università "La Sapienza", Roma, 1992. Extended abstract appeared in: R. Cleaveland (ed.), *Proceedings of CONCUR '92, LNCS 630* , Springer-Verlag. Full version to appear in *Information and Computation.*

[BD94] M. Boreale and R. De Nicola. A symbolic semantics for the π-calculus. Technical Report SI 94 RR 04, Dipartimento di Scienze dell'Informazione Università "La Sapienza", Roma, 1994.

[Dam93] M. Dam. Model checking mobile processes. In E. Best, editor, *Proceedings of CONCUR '93, LNCS 715.* Springer-Verlag, Berlin, 1993.

[FMQ94] G. Ferrari, U. Montanari, and P. Quaglia. The π-calculus with explicit substitutions. Technical report, Università di Pisa, 1994. Accepted to *MFCS'94.*

[Hen91] M. Hennessy. A model for the π-calculus. Technical report, University of Sussex, 1991.

[HL92] M. Hennessy and H. Lin. Symbolic bisimulations. Technical report, University of Sussex, 1992.

[HL93] M. Hennessy and H. Lin. Proof systems for message-passing process algebras. In E. Best, editor, *Proceedings of CONCUR '93, LNCS 715.* Springer-Verlag, Berlin, 1993.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part 1 and 2. *Information and Computation*, 100, 1992.

[PS93] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. Technical report, University of Edinburgh, 1993. To appear in *Information and Computation*.

[San93] D. Sangiorgi. A theory of bisimulation for the π-calculus. In E. Best, editor, *Proceedings of CONCUR '93, LNCS 715*. Springer-Verlag, Berlin, 1993.

[San94] D. Sangiorgi. On the bisimulation proof method. Technical report, University of Edinburgh, 1994. In preparation.

This article was processed using the LaTeX macro package with LLNCS style