Computing Bisimulations for Finite-Control π-Calculus *

Huimin Lin
Laboratory for Computer Science
Institute of Software, Chinese Academy of Sciences
P.O Box 8718, Beijing 100080
E-mail: lhm@ox.ios.ac.cn

Abstract

Symbolic bisimulation avoids the infinite branching problem caused by instantiating input names with all names in the standard definition of bisimulation in the π-calculus. However, it does not automatically lead to an efficient algorithm, because symbolic bisimulation is indexed by conditions on names, and directly manipulating such conditions can be computationally costly. In this paper a new notion of bisimulation is introduced in which the manipulation of maximally consistent conditions is replaced with a systematic employment of schematic names. It is shown that the new notion captures symbolic bisimulation in a precise sense. Based on the new definition an efficient algorithm, which instantiate input names "on-the-fly", is presented to check bisimulations for finite-control π-calculus.

1 Introduction

The aim of this paper is to develop an efficient algorithm for checking bisimulation equivalences for finite-control π-calculus. Unlike in finite state CCS where the definition of bisimulation suggests an effective approach to computing it ([Lar93, FM91]), in the π-calculus the standard definition of bisimulation requires infinite many residuals to be further compared after matching input actions (one for each possible instantiation of the input names), rendering any computation directly based on the definition ineffective. To overcome this "infinite branching" problem the notion of *symbolic bisimulation* has been proposed for the π-calculus [Lin94, Lin95a]. It was adapted from the work on symbolic bisimulation for general value-passing processes [HL95, HL96]. A symbolic bisimulation is a family of binary relations over processes, indexed by conditions (boolean combinations of equality tests over finite sets of names). Instead of instantiating an input name with all names, symbolic bisimulation only requires it to be instantiated with a fresh name for each element of a finite partition of the current indexing condition. A canonical partition of a condition is the set of its *maximally consistent extensions*. Given a condition and

*Supported by the National Natural Science Foundation of China

a finite set of names, its maximally consistent extensions over the set can be effectively enumerated. Thus the definition of symbolic bisimulation implies an algorithm by itself. However, a brute-force enumeration of all maximally consistent extensions of a given condition is both time and space costly (the number of maximally consistent conditions over a set of n names is exponential on n). To improve efficiency, a key observation is that a maximally consistent condition over a name set V amounts to a *partition* of V into equivalence classes. For a condition ϕ which is maximally consistent over V, let us denote by $\mathcal{C}_V(\phi)$ the set of equivalence classes induced by ϕ over V. Then the set of maximally consistent extensions of ϕ over $V \cup \{z\}$, where $z \notin V$, can be obtained by adding z to each equivalence class in $\mathcal{C}_V(\phi)$, one a time, plus a new class $\{z\} \cup \mathcal{C}_V(\phi)$.

Based on this idea we introduce the notion of *schematic bisimulation* where the role of maximally consistent conditions on names are replaced by systematic employment of schematic names. We show that schematic bisimulation defines the same relation as symbolic bisimulation. To compute schematic bisimulation we adapt the "on-the-fly" instantiation algorithm for checking bisimulations for "data-independent processes" ([Lin98]) to the setting of π-calculus. The main strategy is to order all names and, when a new name is required (to instantiate a bound name), always choose the least one which is not used in the current computation.

The rest of the paper is organised as follows. The language of the π-calculus and symbolic bisimulation are introduced in next section. Section 3 motivates schematic bisimulation with a proof that it captures symbolic bisimulation. The algorithm is presented in Section 4. Section 5 outlines how to carry the results obtained so far for late bisimulation to early equivalence. The paper is concluded with Section 6 where related work is also discussed.

2 π-Calculus and Symbolic Bisimulations

The basic entities of the π-calculus are *names*. In this section we first briefly recall some notations and constructions on names, then introduce a version of symbolic bisimulation that suits the purpose of this paper.

2.1 Names and Conditions

We presuppose a countably infinite set $\mathcal{N}$ of *names*, ranging over by $a,\ b,\ x,\ y,\ w, \ldots$, which is totally ordered. *Conditions* are boolean expressions built up by the usual boolean operators applied to name equalities of the form $x = y$. $\neg(x = y)$ will be abbreviated as $x \neq y$. *Name substitutions*, ranged over by σ, are partial functions from $\mathcal{N}$ to $\mathcal{N}$. $[\overline{y}/\overline{x}]$, where $\overline{x}$ is a vector of distinct names, is the name substitution sending $\overline{x}$ to $\overline{y}$. The empty substitution is denoted $\emptyset$. We write $\sigma \models \phi$ if $\phi\sigma = true$, $\phi \Rightarrow \psi$ if $\sigma \models \phi$ implies $\sigma \models \psi$ for any substitution σ, and $\phi = \psi$ for $\phi \Rightarrow \psi$ and $\psi \Rightarrow \phi$ By definition $\sigma\sigma' \models \phi$ if and only if $\sigma' \models \phi\sigma$. It is easy to see that the relation $\phi \Rightarrow \psi$ is decidable.

A condition ϕ is *consistent* if there are no $x,\ y \in \mathcal{N}$ such that $\phi \Rightarrow x = y$ and $\phi \Rightarrow x \neq y$. ϕ is *maximally consistent* on $V \subset \mathcal{N}$ if ϕ is consistent and for any $x,\ y \in V$ either $\phi \Rightarrow x = y$ or $\phi \Rightarrow x \neq y$.

Let the restriction of ϕ on V be $\phi|_V = \bigwedge\{\, x = y \mid x, y \in V,\ \phi \Rightarrow x = y \,\} \wedge \bigwedge\{\, x \neq y \mid x, y \in V,\ \phi \Rightarrow x \neq y \,\}$.

Lemma 2.1 *If ϕ is maximally consistent on V' and $V \subseteq V'$, then $\phi|_V$ is maximally consistent on V.*

Proof: Immediate from the definition of $\phi|_V$. □

Definition 2.2 ψ is a *maximally consistent extension* of ϕ on V, written $\psi \in MCE_V(\phi)$, if

1. $\psi \Rightarrow \phi$,
2. for any e of the form $x = y$ or $x \neq y$ with $n(e) \not\subseteq V$, $\psi \Rightarrow e$ implies $\phi \wedge (\psi|_V) \Rightarrow e$,
3. ψ is maximally consistent on V.

□

If V is finite then $MCE_V(\phi)$ is also finite. $MCE_V(true)$ will be abbreviated as MC_V.

Given a condition ψ, let $\mathcal{C}_\psi$ be the set of equivalence classes determined by ψ, *i.e.* x and y are in the same equivalence class if and only if $\psi \models x = y$. Let σ_ψ be the substitution which maps every name to the least name of the equivalence class it belongs. By definition $\sigma_\psi \models \psi$.

Lemma 2.3 *Suppose $n(\phi) \subseteq V$, $\psi \in MC_V$ and $z \notin V$. then*

1. *$\sigma_\psi \models \phi$ iff $\psi \Rightarrow \phi$.*
2. *$\psi \in MCE_{V \cup \{z\}}(\phi)$ iff*
   - *either $\psi \Leftrightarrow \phi \wedge z = v$ for some $v \in V$*
   - *or $\psi \Leftrightarrow \phi \wedge \bigwedge\{\, z \neq w \mid w \in V \,\}$.*

Proof: As an example we prove *(1)*. The "if" direction is immediate. For the "only if" direction, assuming $\sigma_\psi \models \phi$ and $n(\phi) \subseteq V$, we need to show for any σ if $\sigma \models \psi$ then $\sigma \models \phi$. To this end we prove a stronger result: If $\sigma \models \psi$ then $\sigma \models \phi$ *iff* $\sigma_\psi \models \phi$. Apply induction on the structure of ϕ.

- $\phi \equiv x = y$ where $x, y \in V$. We have

  $$\begin{array}{l} \sigma \models x = y \text{ implies } \psi \Rightarrow x = y \text{ implies } \sigma_\psi \models x = y \\ \sigma \not\models x = y \text{ implies } \psi \not\Rightarrow x = y \text{ implies } \psi \Rightarrow x \neq y \text{ implies } \sigma_\psi \models x \neq y \end{array}$$

  Therefore $\sigma \models x = y$ *iff* $\sigma_\psi \models x = y$.

- $\phi \equiv \neg\phi'$. By the induction hypothesis, $\sigma \models \phi'$ *iff* $\sigma_\psi \models \phi'$. Hence $\sigma \models \neg\phi'$ *iff* $\sigma_\psi \models \neg\phi'$.

$$\text{pre}\ \frac{}{\alpha.t \xrightarrow{\alpha} t} \qquad \text{cond}\ \frac{t \xrightarrow{\alpha} t'}{\phi t \xrightarrow{\alpha} t'}\ \phi = \mathit{true}$$

$$\text{sum}\ \frac{t \xrightarrow{\alpha} t'}{t + u \xrightarrow{\alpha} t'} \qquad \text{par}\ \frac{t \xrightarrow{\alpha} t'}{t \mid u \xrightarrow{\alpha} t' \mid u}\ bn(\alpha) \cap fn(u) = \emptyset$$

$$\text{res}\ \frac{t \xrightarrow{\alpha} t'}{\nu x t \xrightarrow{\alpha} \nu x t'}\ x \notin n(\alpha) \qquad \text{com}\ \frac{t \xrightarrow{a(x)} t' \quad u \xrightarrow{\overline{a}y} u'}{t \mid u \xrightarrow{\tau} t'[y/x] \mid u'}$$

$$\text{open}\ \frac{t \xrightarrow{\overline{a}x} t'}{\nu x t \xrightarrow{\overline{a}(x)} t'}\ a \neq x \qquad \text{close}\ \frac{t \xrightarrow{a(x)} t' \quad u \xrightarrow{\overline{a}(y)} u'}{t \mid u \xrightarrow{\tau} \nu y(t'[y/x] \mid u')}$$

$$\text{id}\ \frac{t[y_1, ..., y_n/x_1, ..., x_n] \xrightarrow{\alpha} t'}{A(y_1, ..., y_n) \xrightarrow{\alpha} t'}\ A(x_1, ..., x_n) \stackrel{def}{=} t$$

Figure 1: π-Calculus Operational Semantics

- $\phi \equiv \phi_1 \wedge \phi_2$. By the induction hypothesis, $\sigma \models \phi_1$ *iff* $\sigma_\psi \models \phi_1$, $\sigma \models \phi_2$ *iff* $\sigma_\psi \models \phi_2$. Hence $\sigma \models \phi_1 \wedge \phi_2$ *iff* $\sigma_\psi \models \phi_1 \wedge \phi_2$

Now let $\sigma \models \psi$. Since $\sigma_\psi \models \phi$ and $n(\phi) \subseteq V$, $\sigma \models \phi$ by the above result. Since σ is arbitrary, $\psi \Rightarrow \phi$. $\square$

2.2 The Language and Symbolic Bisimulation

The language of the π-calculus can be given by the following BNF grammar

$$\begin{array}{rcl} t & ::= & \mathbf{0} \quad | \quad \alpha.t \quad | \quad t + t \quad | \quad \phi t \quad | \quad \nu x t \quad | \quad t \mid t \quad | \quad A(y_1, ..., y_n) \\ \alpha & ::= & \tau \quad | \quad \overline{a}x \quad | \quad a(x) \end{array}$$

We refer to [MPW92] for the intended meaning of these operators, as well as the notions of bound and free names, and that of α-equivalence. We use $fn(t)$ to denote the set of free names of t. We will not distinguish between α-equivalence and syntactical identity, writing $\equiv$ for both. The sets of free and bound names of actions are defined by: $fn(\tau) = bn(\tau) = bn(\overline{a}x) = \emptyset$, $fn(\overline{a}x) = \{a, x\}$, $fn(a(x)) = fn(\overline{a}(x)) = \{a\}$, and $bn(a(x)) = bn(\overline{a}(x)) = \{x\}$. Figure 1 lists the operational semantics rules for this language (we omit the symmetric rules for sum and par). We also disallow the parallel composition operator $\mid$ to appear in the bodies of recursive definitions, and call this restricted language "finite-control" π-calculus.

The symbolic operational semantics is presented in Figure 2, where the symmetric rules for Sum and Par have been omitted. The Res and Open rules use an operation on

$$\text{Pre}\ \frac{}{\alpha.t \xrightarrow{true,\alpha} t} \qquad \text{Cond}\ \frac{t \xrightarrow{\psi,\alpha} t'}{\phi t \xrightarrow{\psi\wedge\phi,\alpha} t'}\ n(\phi)\cap bn(\alpha)=\emptyset$$

$$\text{Sum}\ \frac{t \xrightarrow{\phi,\alpha} t'}{t+u \xrightarrow{\phi,\alpha} t'} \qquad \text{Par}\ \frac{t \xrightarrow{\phi,\alpha} t'}{t \mid u \xrightarrow{\phi,\alpha} t' \mid u}\ bn(\alpha)\cap fn(u)=\emptyset$$

$$\text{Res}\ \frac{t \xrightarrow{\phi,\alpha} t'}{\nu x t \xrightarrow{\nu_x\phi,\alpha} \nu x t'}\ x \notin n(\alpha) \quad \text{Com}\ \frac{t \xrightarrow{\phi,a(x)} t' \quad u \xrightarrow{\psi,\overline{b}y} u'}{t \mid u \xrightarrow{[a=b]\wedge\phi\wedge\psi,\tau} t'[y/x] \mid u'}$$

$$\text{Open}\ \frac{t \xrightarrow{\phi,\overline{a}x} t'}{\nu x t \xrightarrow{\nu_x\phi,\overline{a}(x)} t'}\ x \neq a \quad \text{Close}\ \frac{t \xrightarrow{\phi,a(x)} t' \quad u \xrightarrow{\psi,\overline{b}(y)} u'}{t \mid u \xrightarrow{[a=b]\wedge\phi\wedge\psi,\tau} \nu y(t'[y/x] \mid u')}$$

$$\text{Id}\ \frac{t[y_1,...,y_n/x_1,...,x_n] \xrightarrow{M,\alpha} t'}{A(y_1,...,y_n) \xrightarrow{M,\alpha} t'}\ A(x_1,...,x_n) \stackrel{def}{=} t$$

Figure 2: π-Calculus Symbolic Operational Semantics

conditions ν_x which is defined thus ([BD96]):

$$\begin{array}{ll} \nu_x true = true & \nu_x[x=x] = true \\ \nu_x[x=y] = false & \nu_x[y=z] = [y=z] \\ \nu_x \neg\phi = \neg\nu_x\phi & \nu_x(\phi\wedge\psi) = \nu_x\phi \wedge \nu_x\psi \end{array}$$

Clearly $x \notin n(\nu_x\phi)$.

The relationship between the standard operational semantics and the symbolic version is as expected:

Lemma 2.4 *1. If $t \xrightarrow{\phi,\alpha} u$ and $\psi \Rightarrow \phi$ with $n(\psi) \subseteq fn(t)$ then $t\sigma_\psi \xrightarrow{\alpha\sigma_\psi} u\sigma_\psi$.*

2. If $t\sigma_\psi \xrightarrow{\alpha} p$ then there exist β, u and ϕ such that $\alpha \equiv \beta\sigma_\psi$, $p \equiv u\sigma_\psi$, $\psi \Rightarrow \phi$ and $t \xrightarrow{\phi,\beta} u$.

Proof: By transition induction. □

Definition 2.5 A family of symmetric relations between terms $\mathcal{S} = \{S^\phi\}$, indexed by maximally consistent conditions, is a late symbolic bisimulation if $(t,u) \in S^\phi$ implies $\phi \in MC_{fn(t,u)}$ and

1. If $t \xrightarrow{\psi,\tau} t'$ with $\phi \Rightarrow \psi$ then there is a $u \xrightarrow{\psi',\tau} u'$ such that $\phi \Rightarrow \psi'$ and $(t',u') \in S^{\phi'}$.

2. If $t \xrightarrow{\psi,\overline{a}x} t'$ with $\phi \Rightarrow \psi$ then there is a $u \xrightarrow{\psi',\overline{b}y} u'$ such that $\phi \Rightarrow \psi' \wedge a = b \wedge x = y$ and $(t', u') \in S^{\phi'}$.

3. If $t \xrightarrow{\psi,a(x)} t'$ with $\phi \Rightarrow \psi$ then there is a $u \xrightarrow{\psi',b(y)} u'$ such that $\phi \Rightarrow \psi' \wedge a = b$, and for a fresh name $z \notin fn(t, u)$, for each $\phi' \in MCE_{fn(t,u)\cup\{z\}}(\phi)$, $(t'[z/x], u'[z/y]) \in S^{\phi'}$.

4. If $t \xrightarrow{\psi,\overline{a}(x)} t'$ with $\phi \Rightarrow \psi$ then there is a $u \xrightarrow{\psi',\overline{b}(y)} u'$ such that $\phi \Rightarrow \psi' \wedge a = b$, and for a fresh name $z \notin fn(t, u)$ $(t'[z/x], u'[z/y]) \in S^{\phi'\cup\{\, z\neq w \mid w\in fn(\alpha.t',\beta.u')\,\}}$.

Write $t \sim_L^{\phi} u$ if $(t,\ u) \in S^{\phi} \in \mathcal{S}$ for some late symbolic bisimulation $\mathcal{S}$. $\square$

The above definition differs from the usual definition of symbolic bisimulation (as in [Lin95a, Lin95b], say) in that here it is required each indexing condition be maximally consistent. The difference is only superficial. Let $\sim^{\psi}$ be the symbolic bisimulation relation as defined in [Lin95a, Lin95b], then it can easily be shown that $t \sim^{\psi} u$ if and only if $t \sim_L^{\phi} u$ for every $\phi \in MCE_{fn(t,u)}(\psi)$. $\sim_L$ captures ground bisimulation $\dot{\sim}$ as defined in [MPW92] in the following sense:

Proposition 2.6 *$t \sim_L^{\phi} u$ iff $t\sigma \dot{\sim} u\sigma$ for any $\sigma \models \phi$.*

As a consequence, $t \sim_L^{true} u$ *iff* $t \sim u$ where $\sim$ is the standard (late) bisimulation defined in [MPW92].

3 An Efficient Characterisation

Symbolic bisimulation avoids the infinity involved in the definition of standard bisimulation, caused by instantiating input actions with *all* names, hence it is possible to design an algorithm based on Definition 2.5 to check bisimulation for finite-control π-calculus processes. However, Definition 2.5 uses the construction of maximally consistent extension which in general is expensive to compute. The aim of this section is to formulate an efficient characterisation of symbolic bisimulation, called *schematic bisimulation*.

According to Definition 2.5, when checking if two terms t and u are bisimular over condition ϕ, the set $MCE_{fn(t,u)}(\phi)$ must be computed. Then each element ϕ' in the set is used to compare four kinds of symbolic actions. After matching a free action (τ or free output) ϕ' serves as the new indexing condition for the residual terms t' and u'. Note that ϕ' is maximally consistent on $fn(t', u')$ because $fn(t', u') \subseteq fn(t, u)$, hence, up-to logical equivalence, is the *only* element of $MCE_{fn(t',u')}(\phi')$. For a bounded output action with bound name x, the new indexing condition for t' and u' is $\phi'\cup\{\, z \neq w \mid w \in fn(\alpha.t', \beta.u')\,\}$, and it is easy to see that this condition is also maximally consistent on $fn(t', u')$ because ϕ' is maximally consistent on $fn(t, u)$ and $fn(t', u') \subseteq fn(t, u) \cup \{z\}$. The only place where such maximal consistence property is not preserved is the input clause: here we need to compute $MCE_{fn(t,u)\cup\{z\}}(\phi)$ since ϕ is only maximally consistent on $fn(t, u)$. So the key to an efficient bisimulation algorithm lies on finding an efficient way to enumerating representatives of $MCE_{fn(t,u)\cup\{z\}}(\phi)$ with ϕ being maximally consistent on $fn(t, u)$.

Observe that a maximal consistent condition on a set of names corresponds to a *partition* of the set into equivalent classes. All partitions of $fn(t, u) \cup \{z\}$ which are extensions of a given partition $\mathcal{P}$ of $\mathit{fn}(t,u)$ can be obtained by adding z, in turn, to each equivalence class in $\mathcal{P}$, plus a "new" partition formed by adding $\{z\}$ into $\mathcal{P}$. This process amounts to "instantiate" the input name with a representative from each equivalence class in $\mathcal{P}$, plus a fresh name. If we restrict to symbolic bisimulations in which every indexing condition is maximally consistent on the free names of the indexed terms, and keep consistently this instantiation strategy for every input actions, then indeed the free names in $\mathit{fn}(t,u)$ are in one-to-one correspondence with the equivalence classes on $\mathit{fn}(t,u)$ induced by the indexing condition, hence uniquely determine the condition. As a consequence we do not need to keep the indexing conditions.

With this intuition in mind we now define a new version of bisimulation, called *schematic bisimulation*, between processes.

Definition 3.1 A symmetric relation R between processes is a schematic bisimulation if $(p,q) \in R$ implies:

- If $p \xrightarrow{a(x)} p'$ then $q \xrightarrow{a(y)} q'$ for some q' and
  - for each $v \in \mathit{fn}(p,q)$, $(p'[v/x], q'[v/y]) \in R$.
  - for a fresh $z \notin \mathit{fn}(p,q)$, $(p'[z/x], q'[z/y]) \in R$.
- If $p \xrightarrow{\overline{a}(x)} p'$ then $q \xrightarrow{\overline{a}(y)} q'$ for some q' and for a fresh $z \notin \mathit{fn}(p,q)$, $(p'[z/x], q'[z/y]) \in R$.
- If $p \xrightarrow{\alpha} p'$ for any other action α then $q \xrightarrow{\alpha} q'$ for some q' and $(p',q') \in R$.

Write $p \asymp q$ if there exists a schematic bisimulation R s.t. $(p,q) \in R$. □

Theorem 3.2 *For any $\phi \in MC_{\mathit{fn}(t,u)}$, $t \sim_L^{\phi} u$ if and only if $t\sigma_\phi \asymp u\sigma_\phi$.*

Proof: ($\Longrightarrow$) Suppose $(t,u) \in S^\phi \in \mathcal{S}$ for some symbolic bisimulation $\mathcal{S}$. Define

$$R = \{\, (t\sigma_\phi, u\sigma_\phi) \mid (t,u) \in S^\phi, \phi \in MC_{\mathit{fn}(t,u)} \text{ for some } S^\phi \in \mathcal{S} \,\}$$

We show R is a schematic bisimulation.

Let $(t\sigma_\phi, u\sigma_\phi) \in R$ and $t\sigma_\phi \xrightarrow{\alpha} p$. There are four cases to consider, according to the possible types of α. Here we only examine the hardest case when $\alpha \equiv a(x)$. Without lose of generality we may assume $x \notin \mathit{fn}(t)$.

By Lemma 2.4, there exist a', t' and ϕ' such that $\alpha = a'\sigma_\phi$, $p \equiv t'\sigma_\phi$, $\phi \Rightarrow \phi'$ and $t \xrightarrow{\phi', a'(x)}_L t'$. Since $(t,u) \in S^\phi$ and $\phi \in MC_{\mathit{fn}(t,u)}$, $u \xrightarrow{\psi', b'(y)}_L u'$ for some u' such that $\phi \Rightarrow \psi' \wedge a' = b'$ and for a fresh $z \notin \mathit{fn}(t,u)$, for each $\psi \in MCE_{\mathit{fn}(t,u)\cup\{z\}}(\phi)$, $(t'[z/x], u'[z/y]) \in S^\psi$. We may assume z is the least name which is larger than any name in $\mathit{fn}(t,u)$. By Lemma 2.4 $u\sigma_\phi \xrightarrow{b(y)} u'\sigma_\phi$ with $b = b'\sigma_\phi$. Since $\phi \Rightarrow a' = b'$, $a'\sigma_\phi = b'\sigma_\phi$, hence $b = a$. Furthermore

- For each $v \in \mathit{fn}(t\sigma_\phi, u\sigma_\phi)$, by Lemma 2.3, $\phi \wedge v = z \in MCE_{\mathit{fn}(t,u)\cup\{z\}}(\phi)$. So $(t'[z/x], u'[z/y]) \in S^{\phi\wedge v=z}$. Therefore $(t'[z/x]\sigma_{\phi\wedge v=z}, u'[z/y]\sigma_{\phi\wedge v=z}) \in R$. Since $t'\sigma_\phi[v/x] \equiv t'[z/x]\sigma_{\phi\wedge v=z}$ and $u'\sigma_\phi[v/y] \equiv u'[z/y]\sigma_{\phi\wedge v=z}$, $(t'\sigma_\phi[v/x], u'\sigma_\phi[v/y]) \in R$.

- For the fresh name z, let

$$\psi = \phi \wedge \bigwedge \{ z \neq w \mid w \in \mathit{fn}(t,u) \}$$

By Lemma 2.3, $\psi \in MCE_{\mathit{fn}(t,u)\cup\{z\}}(\phi)$. By the definition of R, $(t'[z/x]\sigma_\psi, u'[z/y]\sigma_\psi) \in R$. By the definition of ψ, $\sigma_\psi = \sigma_\phi \cup \{z \mapsto z\}$. Hence $t'\sigma_\phi[z/x] \equiv t'[z/x]\sigma_\psi$, $u'\sigma_\phi[z/y] \equiv u'[z/y]\sigma_\psi$. Therefore $(t'\sigma_\phi[z/x], u'\sigma_\phi[z/y] \in R$.

($\Longleftarrow$) Suppose $(t\sigma_\psi, u\sigma_\psi) \in R$ for some schematic bisimulation R. Define

$$S^\phi = \{ (t,u) \mid (t\sigma_\phi, u\sigma_\phi) \in R \text{ for some } \phi \in MC_{\mathit{fn}(t,u)} \}$$

and $\mathcal{S} = \{S^\phi\}$. We show $\mathcal{S}$ is a symbolic bisimulation.

Let $(t,u) \in S^\phi \in \mathcal{S}$ and $t \xrightarrow{\phi',\alpha}_L t'$ with $\phi \Rightarrow \phi'$. Again we only consider the input case, $\alpha \equiv a(x)$. By Lemma 2.4, $t\sigma_\phi \xrightarrow{a'(x)} t'\sigma_\phi$ where $a' = a\sigma_\phi$. So $u\sigma_\phi \xrightarrow{a'\sigma_\phi} q$ for some q with the properties as specified in Definition 3.1. By Lemma =2.4, there exist b, u' and ψ' such that $q \equiv u'\sigma_\phi$, $a' = b\sigma_\phi$ and $u \xrightarrow{\psi',b(y)}_L u'$. Let z be the least name which is larger than any name in $\mathit{fn}(t,u)$. By Lemma 2.3, for each $\phi'' \in MCE_{\mathit{fn}(t,u)\cup\{z\}}(\phi)$, there are two possibilities:

- $\phi'' \Leftrightarrow \phi \wedge z = w$ for some $w \in \mathit{fn}(t,u)$. Let $v = \sigma_\phi(w)$. Then $v \in \mathit{fn}(t\sigma_\phi, u\sigma_\phi)$, $\phi \Rightarrow v = w$ and $\phi'' \Leftrightarrow \phi \wedge z = v$. Also $(t'\sigma_\phi[v/x], u'\sigma_\phi[v/y]) \in R$. Note that $\sigma_{\phi''}(z) = v$. Hence $t'[z/x]\sigma_{\phi''} \equiv t'\sigma_\phi[v/x]$, $u'[z/y]\sigma_{\phi''} \equiv u'\sigma_\phi[v/y]$. Therefore $(t'[z/x]\sigma_{\phi''}, u'[z/y]\sigma_{\phi''}) \in R$. By the definition of $\mathcal{S}$, $(t'[z/x], u'[z/y]) \in S^{\phi''}$.

- $\phi'' \Leftrightarrow \phi \wedge \bigwedge\{ z \neq w \mid w \in \mathit{fn}(t,u) \}$. In this case, $\sigma_{\phi''} = \sigma_\phi \cup \{z \mapsto z\}$ and $(t'\sigma_\phi[z/x], u'\sigma_\phi[z/y]) \in R$. Furthermore, $t'[z/x]\sigma_{\phi''} \equiv t'\sigma_\phi[z/x]$, $u'[z/y]\sigma_{\phi''} \equiv u'\sigma_\phi[z/y]$. Hence $(t'[z/x]\sigma_{\phi''}, u'[z/y]\sigma_{\phi''}) \in R$. By the definition of $\mathcal{S}$, $(t'[z/x], u'[z/y]) \in S^{\phi''}$.

Therefore $\mathcal{S}$ is a symbolic bisimulation. □

4 An "On-the-fly" Algorithm

The definition of schematic bisimulation only requires to instantiate each input name with the free names of the current processes, plus *one* fresh name, thus paving the way to an efficient bisimulation checking algorithm.

The algorithm presented in Figure 3 is adapted from the "on-the-fly" algorithm for value-passing processes ([Lin98]). It combines the well-known on-the-fly bisimulation algorithm for standard labelled transition systems with the strategy of instantiating input names "on-the-fly". It assumes a countably infinite subset $\mathcal{SN} \subset \mathcal{N}$ which is totally ordered. The function *nextSN*(s, t) returns the smallest name in $\mathcal{SN}$ that does not appear in the set of free names at states s and t.

The function $\mathit{bisim}(p,q)$ starts with the initial pair (p,q), trying to find the smallest bisimulation containing the pair by matching transitions from them. While traveling the

$bisim(p,\ q) =\ \{$

$\quad NotBisim := \{\ \}$

$\quad$ fun $bis(p,\ q) =\ \{$

$\qquad Visited$, Assumed $= \{\ \ \}, \{\ \ \}$

$\qquad match(p,\ q)$

$\qquad \}$

$\quad \}$ handle $\ WrongAssumption \Rightarrow bis$(p, q)

$\quad$ return$(bis(p,\ q))$

$match(p,\ q) =$

$\quad Visited := \{(p,\ q)\} \cup Visited$

$\quad$ b $= \bigwedge_{\alpha \in \{\tau, \overline{a}x, \overline{a}, a\}} match_{\alpha}(p,\ q)$

$\quad$ if b = false then $\{$

$\qquad NotBisim := \{(p,\ q)\} \cup NotBisim$

$\qquad$ if $(p,\ q) \in Assumed$ then raise $WrongAssumption\}$

$\quad$ return(b)

$match_{\{\ \alpha | \alpha \in \{\tau, \overline{a}x\}\ \}}(p,\ q) =$

$\quad$ for each $s \xrightarrow{\alpha} p_i,\ t \xrightarrow{\alpha} q_j$

$\qquad b_{ij} = close(p_i,\ q_j)$

$\quad$ return$((\bigwedge_i (\bigvee_j b_{ij})) \wedge (\bigwedge_j (\bigvee_i b_{ij})))$

$match_{\{a(x)\}}(p,\ q) =$

$\quad$ for each $s \xrightarrow{a(x)} p_i,\ t \xrightarrow{a(y)} q_j$

$\qquad$ for each $v \in fn(p, q) \cup \{nextSN(p, q)\}$

$\qquad\qquad b_{ij}^{v} = close(p_i[v/x],\ q_j[v/y])$

$\qquad b_{ij} = \bigwedge_{v \in fn(p,q)} b_{ij}^{v}$

$\quad$ return$((\bigwedge_i (\bigvee_j b_{ij})) \wedge (\bigwedge_j (\bigvee_i b_{ij})))$

$match_{\overline{a}}(p,\ q) =$

$\quad$ for each $s \xrightarrow{a(x)} p_i,\ t \xrightarrow{a(y)} q_j$

$\qquad$ for $z = nextSN(p, q)$

$\qquad b_{ij} = close(p_i[z/x],\ q_j[z/y])$

$\quad$ return$((\bigwedge_i (\bigvee_j b_{ij})) \wedge (\bigwedge_j (\bigvee_i b_{ij})))$

$close(p,\ q) =$

$\quad$ if $(p,\ q) \in NotBisim$ then return$(false)$

$\quad$ else if $(p,\ q) \in Visited$ then $\{$

$\qquad Assumed := \{(p,\ q)\} \cup Assumed$

$\qquad$ return$(true)$

$\quad \}$

$\quad$ else return$(match(p,\ q))$

Figure 3: The “on-the-fly” Algorithm

symbolic graphs, at each pair of nodes the algorithm uses the values for the state variables to evaluate the edges, according to the operational semantics in Figure 1, producing the outgoing transitions and next states. The transitions are then matched for bisimulation, and the algorithm goes on to the new state pairs if the matches are successful. However, when input edges are encountered the evaluation can not be fully carried out because the values for the input names at the next pair of nodes are not yet known. In this case the algorithm takes each free names of the current processes, one by one, plus a fresh name, to instantiate the input names "on-the-fly", then moves to the new states.

The core function, *match*, performs a depth-first travel on the product of the two transition graphs which are never fully created. Instead states and transitions of the concrete graphs are only generated from the symbolic graphs piece-by-piece, when needed. Whenever a new pair of states are encountered it is stored in *Visited*. If two states fail to match each other's transitions then they are not bisimilar and the pair is inserted into *NotBisim*. In case the current state pair has been visited before, we first check if it is in *NotBisim* and return *false* if so; Otherwise a loop has been detected and we make the assumption that the two state are bisimilar, by inserting the pair into *Assumed*, and return *true*. If we find the two states are not bisimilar after finishing searching the loop, then we know that the assumption is wrong, so we first add the pair into *NotBisim* and then raise the exception *WrongAssumption* which forces a rerun of *match*, with one piece of new information, namely the two states in this pair are not bisimilar.

The correctness of the algorithm is not difficult to justify. Each call of *bis*(s,t) performs a depth-first search in the product graph of the two concrete transition graphs (which are created piece by piece from the symbolic graphs during the search). *bis*(s,t) is only recalled when the exception *WrongAssumption* has been raised, in this case the size of *NotBisim* has been increased by at least one. This ensures that *bis*(s,t) can only be called for finitely many times. Therefore *bisim*(s,t) will always terminate. If *bisim*(s,t) terminates with *true*, then the set (*Visited* $-$ *NotBisim*) constitutes a schematic bisimulation containing (s,t); If it terminates with *false* then (s,t) is in *NotBisim* and they are not bisimilar. So we have the following theorem:

Theorem 4.1 *For finite symbolic graphs G and H with roots n and m and initial values σ and σ', respectively, bisim$((n,\sigma),(m,\sigma'))$ always terminates, and it returns true if and only if $(n,\sigma) \asymp (m,\sigma')$.*

To see the complexity of the algorithm, let N be the number of nodes in the production graph of the two transition graphs. The time required for the first call of *bis* is (at most) $O(N)$, for the second call is $O(N-1)$, ... Hence the worst case complexity of *bisim* is $O(N+(N-1)+...+1) = O(N^2)$.

5 Early Bisimulations

So far we have concentrated on late bisimulation. It is straightforward to adapt the results to early bisimulation. The definitions of early symbolic and early schematic bisimulations can be obtained by changing the input clause of Definitions 2.5 and 3.1, respectively:

The input case of early symbolic bisimulation:

If $t \overset{\psi,a(x)}{\longrightarrow} t'$ with $\phi \Rightarrow \psi$ then for a fresh name $z \notin fn(t,u)$, for each $\phi' \in MCE_{fn(t,u)\cup\{z\}}(\phi)$, there is a $u \overset{\psi',b(y)}{\longrightarrow} u'$ such that $\phi \Rightarrow \psi' \wedge a = b$ and $(t'[z/x], u'[z/y]) \in S^{\phi'}$.

The input case of early schematic bisimulation:

If $p \overset{a(x)}{\longrightarrow} p'$ then

- for each $v \in fn(p,q)$, there is a q' such that $q \overset{a(y)}{\longrightarrow} q'$ and $(p'[v/x], q'[v/y]) \in R$.

- for a fresh $z \notin fn(p,q)$, there is a q' such that $q \overset{a(y)}{\longrightarrow} q'$ and $(p'[z/x], q'[z/y]) \in R$.

Then Theorem 3.2 carries over to the early case.
For the algorithm, replace the function $match_{\{a(x)\}}(p,\ q)$ with:

$$
\begin{array}{l}
match^E_{\{a(x)\}}(p,\ q) = \\
\qquad \text{for each } v \in fn(p,q) \cup \{nextSN(p,q)\} \\
\qquad\qquad \text{for each } s \overset{a(x)}{\longrightarrow} p_i,\ t \overset{a(y)}{\longrightarrow} q_j \\
\qquad\qquad\qquad b^v_{ij} = close(p_i[v/x],\ q_j[v/y]) \\
\qquad\qquad b_{ij} = \bigwedge_{v \in fn(p,q)} b^v_{ij} \\
\qquad \text{return}((\bigwedge_i(\bigvee_j b_{ij})) \wedge (\bigwedge_j(\bigvee_i b_{ij})))
\end{array}
$$

6 Conclusion

We have present an implementation-oriented definition of bisimulation for the π-calculus. It captures symbolic bisimulation (hence concrete bisimulation as well). The improvement in efficiency is gained by replacing the explicit manipulation of maximally consistent conditions in the symbolic bisimulation definition with a systematic way to choosing representatives for equivalence classes of names. The new definition leads naturally to an algorithm for computing bisimulation which employs an "on-the-fly" strategy to instantiate input names.

Related Work The *Mobility Workbench* [VM94] implemented an "on-the-fly" algorithm for open bisimulation which is a finer relation than late and early bisimulations, as names need to be instantiated when making *any moves*, not just the input ones. [LC98] adapted an algorithm proposed in [Lin96] for value-passing processes to π-calculus, to reduce the problem of checking both late and early bisimulations to finding the maximal solutions of certain *predicate equation systems* over names. Although such solutions can be effectively computed (using the ideas presented in this paper, for instance), due to the simplicity of the value domain in π-calculus (just a plain set of names), this possibility was not pursued there. The use of the least name not present in the current computation state in the "on-the-fly" algorithm bears similarity to the *least number heuristic* of [Zha94] which has gained success in the finite model generation approach to theorem proving.

References

[BD96] Michele Boreale and Rocco De Nicola. A symbolic semantics for the π-calculus. *Information and Computation*, 126(1):34–52, 1996.

[FM91] J.-C. Fernandez and L. Mounier. On the fly verification of behavioural equivalences and preorders. In *CAV'91*, volume 575 of *Lecture Notes in Computer Science*, pages 181–191. Springer–Verlag, 1991.

[HL95] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.

[HL96] M. Hennessy and H. Lin. Proof systems for message-passing process algebras. *Formal Aspects of Computing*, 8:408–427, 1996.

[Lar93] K.G. Larsen. Efficient local correctness checking (extended abstract). In *Proceedings of 4th International Workshop on Computer Aided Verification, CAV'92*, volume 663 of *Lecture Notes in Computer Science*, pages 30–43. Springer–Verlag, 1993.

[LC98] Z. Li and H. Chen. Cheking strong/weak bisimulation equivalences and observation congruence for the π-calculus. In *ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, pages 707–718. Springer–Verlag, 1998.

[Lin94] H. Lin. Symbolic bisimulations and proof systems for the π-calculus. Report 7/94, Computer Science, University of Sussex, 1994.

[Lin95a] H. Lin. Complete inference systems for weak bisimulation equivalences in the π-calculus. In *TAPSOFT'95*, volume 915 of *Lecture Notes in Computer Science*. Springer–Verlag, 1995.

[Lin95b] H. Lin. Unique fixpoint induction for the π-calculus. In *CONCUR'95*, volume 962 of *Lecture Notes in Computer Science*. Springer–Verlag, 1995.

[Lin96] H. Lin. Symbolic transition graph with assignment. In *CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 50 – 65. Springer–Verlag, 1996.

[Lin98] H. Lin. "on-the-fly instantiation" of value-passing processes. In *FORTE/PSTV'98*, pages 215–230. Kluwer Academic Publishers, 1998.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile proceses, part I,II. *Information and Computation*, 100:1–77, 1992.

[VM94] B. Vicor and F. Moller. The mobility workbench – a tool for the π-calculus. In *Proc. CAV'94*, Lecture Notes in Computer Science, pages 428–440. Springer–Verlag, 1994.

[Zha94] J. Zhang. *The Generation and Applications of Finite Models*. Ph.d. thesis, Institute of Software, Chinese Academy of Sciences, 1994.