A Partition Refinement Algorithm for the π-calculus[*]

Marco Pistore[1] and Davide Sangiorgi[2]

[1] Department of Computer Science, University of Pisa, Italy
[2] INRIA - Sophia Antipolis, France.

Abstract. The *partition refinement algorithm* [10, 6] is the basis for most of the tools for checking bisimulation equivalences and for computing minimal realisations of CCS-like finite state processes.
In this paper, we present a partition refinement algorithm for the π-calculus [7], a development of CCS where channel names can be communicated. It can be used to check bisimilarity and to compute minimal realisations of finite control processes — the π-calculus counterpart of CCS finite state processes. The algorithm is developed for *strong open bisimulation*, and can be adapted to *late* and *early* bisimulations, as well as to *weak* bisimulations. To arrive at the algorithm, a few laws and four characterisations of open bisimulation are proved.

1 Introduction

Bisimulation is widely used for defining behavioural equivalences on terms of process description languages. It has been extensively studied in CCS, where efficient algorithms and tools for bisimulation checking have been devised. A prominent role among these algorithms is occupied by the *partition refinement algorithm* [10, 6]. It works in two phases: In the first, the state spaces of the processes to be checked (i.e., the set of their derivatives under arbitrary sequences of actions) are separately generated; in the second, a partition refinement procedure is applied to the union of the state spaces, which terminates when the equivalence classes of bisimulation are found. Most important, the same algorithm can be used to obtain a *minimal realisation* of a process P, i.e., a process which has the minimum number of states and transitions among all those bisimilar with P. In the case of CCS, algorithms for bisimilarity generally apply only to *finite-state* processes, syntactically described by disallowing parallel composition within recursive definitions.

In this paper, we study bisimulation checking in the π-calculus, a development of CCS where channel names can be communicated. Name-passing increases the expressiveness of the calculus, but it also dramatically affects the theory — above all the definition of bisimulation and its associated algorithms.

---

[*] Research supported by CNR project “Strumenti per la Verifica di Proprietà Critiche di Sistemi Concorrenti e Distribuiti” and by CNET project “Modélisation de Systèmes Mobiles”.

In the π-calculus, the syntactic counterpart of CCS finite state processes are the *finite control* processes [3]. Due to the creation of new names, finite control processes can exhibit an infinite-state behaviour.

Three definitions of bisimulation, called *late* [7], *early* [7] and *open* [11], have been proposed for the π-calculus, and vary in the way name instantiations are handled. Here, we focus on open bisimulation, for three main reasons: First, by contrast with the other two, open bisimulation is a full congruence; this can be used, for instance, for compositional minimisations of processes. Secondly, the average complexity of checking open bisimulation is expected to be substantially lower than those of late and early bisimulations [11]. Thirdly, a partition refinement algorithm for open bisimulation presents more difficulties, hence one expects to extract algorithms for late and early bisimulations from it.

Open bisimulation. Differently from early and late bisimulations, where free names of processes are viewed as constants (hence cannot be identified), in open bisimulation free names are viewed as free variables. However, permanent inequalities on names can be imposed by means of *distinctions*, i.e., irreflexive and symmetric relations on names. Distinctions allow us, for instance, to create constants by declaring certain names different from all other names; they are also useful to handle name extrusions — see below. We shall write $\sim_D$ to denote open bisimilarity under distinction D.

We review some aspects of the symbolic characterisation of open bisimulation [11]. Symbolic transitions are of the form $P \xrightarrow{M\,,\,\alpha} P'$ where, intuitively, M represents the least condition, in the form of a conjunction of equalities between names, under which action α can occur. A condition M determines an equivalence relation on names; chosen a representative for each equivalence class, M also determines a name substitution σ_M which maps names to their representatives. We write $\emptyset$ for the "true" condition, that is the empty conjunction. Among the possible forms of action α, there is the *bound output* $\overline{a}(b)$, which denotes the emission at a of the *private* name b. Approximately, the clause of bisimilarity on bound outputs says that if $P \sim_D Q$, and $\mathrm{fn}(P, Q)$ is the set of free names in P and Q, then:

$$\begin{array}{l}\text{whenever } P \xrightarrow{M\,,\,\overline{a}(b)} P' \text{ with } b \text{ not free in } Q\text{, there are } N \text{ and } Q' \text{ s.t.}\\ Q \xrightarrow{N\,,\,\overline{a}(b)} Q',\quad M \text{ implies } N,\quad \text{and}\\ P' \;\sim_{D'}\; Q'\sigma_M,\quad \text{for } D' \overset{\text{def}}{=} D\sigma_M \cup \{b\} \times \mathrm{fn}(P, Q)\end{array} \tag{1}$$

Derivative Q' and distinction D are updated according to the name equalities imposed by condition M, and distinction $\{b\} \times \mathrm{fn}(P, Q)$ is added to record the creation of the new name b, which cannot be identified with the previously known names.

Main problems for a partition algorithm. In (1), three forms of dependencies between P and Q affect the transitions and the derivatives to examine:

Dep.1: The name emitted by P cannot occur free in Q;

Dep.2: The condition M in the transition of P determines a substitution σ_M which is applied onto the derivative of Q;

Dep.3: There is a global distinction, which is updated using informations (the free names) from both processes.

These dependencies prevent us from generating the state spaces of P and Q separately from each other, as requested in the first phase of the partition refinement algorithm. Each dependency introduces a separate problem, which we now discuss in more detail.

Dep.1 is imposed to ensure that the bound name chosen by P can also be chosen by Q, and is necessary because bisimilar processes may have different sets of free names, like

$$P_1 \overset{\text{def}}{=} P_2 + [c = e]\overline{a}(b).\, c(d).\, \mathbf{0} \qquad \text{and} \qquad P_2 \overset{\text{def}}{=} \overline{a}(b).\, e(d).\, \mathbf{0} \tag{2}$$

(a matching $[c = e]$ means "if $c = e$ then"). The choice of the bound name could be made locally if *active names* [8], instead of free names, were used. A name x is *active* for a process P if x does affect the behaviour of P, i.e., P and $\boldsymbol{\nu} x\, P$ are not bisimilar ($\boldsymbol{\nu}\_$ is the restriction construct). Active names would eliminate Dep.1 because bisimilar processes have the same sets of active names. Unfortunately, computing active names is as difficult as computing bisimilarity. For instance, x is active in $a(y).\, ([y = x]P + Q)$ iff $P\{y/x\}$ and $Q\{y/x\}$ are bisimilar.

Dep.2 arises because conditions M and N on the transitions from P and Q may be logically non-equivalent. This situation happens, for instance, when comparing processes P_1 and P_2 in (2), where transition $P_1 \xrightarrow{[c=e],\, \overline{a}(b)} c(d).\, \mathbf{0}$ is matched by transition $P_2 \xrightarrow{\emptyset,\, \overline{a}(b)} e(d).\, \mathbf{0}$, for under the condition $[c = e]$ the two derivatives are bisimilar (in this case, they are actually equal). However, this transition of P_1 does not add anything interesting to its behaviour, since covered by transition $P_1 \xrightarrow{\emptyset,\, \overline{a}(b)} e(d).\, \mathbf{0}$, which has a logically weaker condition. The first transition can therefore be regarded as *redundant*. Indeed, if redundant transitions were ignored in the bisimulation clauses, then Dep.2 could be removed. But again, determining whether a transition is non-redundant is as difficult as computing bisimilarity.

To avoid Dep.3, distinctions should be made local to processes, and should be locally updated. But the update of the local distinction of a process P has to depend on its free names, otherwise the update might not be sound. As a consequence, since bisimilar processes may have different free names, one must be able to compare processes with different local distinctions. This makes it hard to obtain a *transitive* bisimulation relation and to recover open bisimulation.

Our approach. In general, computing bisimilarity, active names or non-redundant transitions is of equal difficulty, since, unfortunately, they all depend from each other. Our algorithm will hence compute bisimilarity, active names and non-redundat transitions *at the same time*. Since bisimulation is defined as a maximal fixed-point, whereas active names and non-redundant transitions are defined as minimal fixed-points, the algorithm approximates bisimulation from above, and active names and non-redundant transitions from below. At the beginning, all processes are assumed bisimilar, no name is assumed active and no transition is assumed non-redundant. In each approximation step, the appropriate transitions and derivatives of processes are selected according to the current estimation of active names and non-redundant transitions, and the standard partition refinement algorithm is applied. At the end, active names and non-redundant transitions are updated. In this way, at each step the assumed set of bisimilar processes decreases, whereas the assumed sets of active names and non-redundant transitions increase. This procedure is repeated until a fixed-point is reached.

The update of the distinction which is local to a process P uses its free names. The bisimulation relation computed by the algorithm, and defined on processes with a local distinction, is not transitive; but it enjoys a weak form of transitivity which is enough to prove a characterisation theorem w.r.t. open bisimulation and to apply the partition refinement algorithm (in which transitivity is important).

Our algorithm can be used on finite-control processes to check open bisimulation and to compute minimal realisations of processes.

Related work. The closest work to ours is [8], where a partition refinement algorithm for early and late bisimulation is obtained which works for finite-control processes without the matching operator. [8] has inspired our work, and has provided us with useful insights. However, our technical development is quite different, because none of the problems in Dep.1–3 arise in [8]. Dep.2 and Dep.3 do not arise because they are specific to open bisimulation; Dep.1 does not pose a serious obstacle because, in late and early bisimulations without matching, active names of processes are trivial to compute: Roughly, a name x is active in a process P if x appears as a free name in a label of a computation of P. Hence active names can be computed by a standard transitive closure procedure.

The *Mobility Workbench* [12] is a tool for mechanically checking open bisimulation on finite control processes. It adopts an *on the fly* [4] approach, as opposed to the partitioning approach (in on-the-fly, the state spaces of processes compared are created at the same time as the candidate bisimulation relation). A disadvantage of on-the-fly is that it cannot be used to give the minimal realisation of a process. Moreover, due to the need of backtracking, in general on-the-fly is less efficient than partitioning both in time and in space (especially in the case of weak bisimulations). However, on-the-fly can be superior on processes which exhibit a limited degree of non-determinism; and it may return an answer even on non-finite-control processes, if not bisimilar.

In [3] decidability of early and late bisimilarity for finite control processes is proved. In particular, it is shown that for every pair of finite control processes only a finite number of names is sufficient for checking bisimilarity; once the number of names if guessed, the state space of both agents can be built and a partition refinement algorithm can be applied. However, this approach can be expensive and, since the number of names can be guessed only for pairs of processes, it does not provide us with minimal realisations. Open bisimulation, and hence Dep.2 and Dep.3, are not considered in [3].

For lack of space, in this short version some technical definitions and all proofs have been suppressed.

2 π-calculus

We briefly review the syntax of the π-calculus, and the definition of open bisimulation. Letters $a, b, \ldots, x, y, \ldots$ range over the infinite ordered set $\mathcal{N}$ of names, and K over the set of process identifiers. The class of processes is built from the operators of inaction, prefixing, matching, parallel composition, restriction, sum, and recursion; a prefix can be a silent prefix, an input, a free output, or a bound output:

$$\begin{aligned} P, Q &:= \mathbf{0} \;\big|\; \alpha.\, P \;\big|\; [a\!=\!b]P \;\big|\; P \mid Q \;\big|\; \boldsymbol{\nu} a\, P \;\big|\; P + Q \;\big|\; K\langle \tilde{a} \rangle \\ \alpha &:= \tau \;\big|\; a(b) \;\big|\; \overline{a}b \;\big|\; \overline{a}(b). \end{aligned}$$

Each identifier K has an associated arity and a definition of the form $K \overset{\text{def}}{=} (\tilde{a})P$; in $K\langle\tilde{a}\rangle$ and $(\tilde{a})P$, tuple $\tilde{a}$ is of the same length as the arity of K. We give sum and parallel composition the lowest syntactic precedence among the operators. In $a(b).\,P$, $\boldsymbol{\nu} b\, P$, and $\overline{a}(b).\,P$, all free occurrences of name b in P are bound. *Free names* (fn), *bound names* (bn) and *names* (n) of processes and prefixes, name substitutions, and alpha conversion are defined as expected. The *extruded names* of an action α, written $\mathrm{en}(\alpha)$, are its bound names if α is a bound output, are the empty set otherwise. We use σ to range over substitutions. Application of a substitution σ to a process P and to an action α are written $P\sigma$ and $\alpha\sigma$, respectively (in $\alpha\sigma$, the bound names of α are not touched).

Conditions are finite conjunctions of matching, like $[a\!=\!b][c\!=\!d]$. We use M and N to range over conditions, and $\mathrm{n}(M)$ for the names which appear in M. We write $M \rhd N$ if M implies N, i.e., the equalities in M imply those in N; $M \lhd\!\rhd N$ is a shortcut for "$M \rhd N$ and $N \rhd M$", whereas $M \not\lhd\!\rhd N$ is a shortcut for "$M \rhd N$ but not $N \rhd M$". Notice that, since names are ordered, there are canonical forms for conditions, which are unique up to $\lhd\!\rhd$. Given a matching M, we denote by σ_M the substitution which selects the minimal representative out of each equivalence class on names induced by M, i.e., $\sigma_M(a) = \min\{b \mid M \rhd [a\!=\!b]\}$.

A *distinction* is a finite symmetric irreflexive relation on $\mathcal{N}$, which expresses permanent inequalities on names, i.e., if (a, b) is in the distinction, then a must be kept separate from b. We use D to range over distinctions and $\mathrm{n}(D)$ for the names which are mentioned in D. A substitution σ *respects* a distinction D if $(a, b) \in D$ implies $\sigma(a) \neq \sigma(b)$; in this case we write $D\sigma$ for the distinction $\{(\sigma(a), \sigma(b)) \mid (a, b) \in D\}$. Similarly, a matching M *respects* a distinction D if σ_M respects D. Sometimes, in the expressions defining distinctions we shall avoid to give all symmetric pairs. If N is a set of names, then $D - N$ is the distinction $\{(a, b) \in D \mid a, b \notin N\}$ and $D \cap N$ is the distinction $\{(a, b) \in D \mid a, b \in N\}$.

The symbolic transition system for open bisimulation [11] is reported in Table 1 (the symmetric of `sum`, `par`, `com` and `close` have been omitted). Transitions have the form $P \overset{M,\,\alpha}{\rightsquigarrow} P'$, where M represents the minimal condition on names required by P to perform that action.

$$\texttt{alpha:}\quad \frac{P' \overset{\mu}{\rightsquigarrow} P'' \qquad P \text{ and } P' \text{ alpha convertible}}{P \overset{\mu}{\rightsquigarrow} P''}$$

$$\texttt{pre:}\ \alpha.\, P \overset{\emptyset\,,\,\alpha}{\rightsquigarrow} P \qquad\qquad \texttt{con:}\ \frac{P\{\tilde{a}/\tilde{b}\} \overset{\mu}{\rightsquigarrow} P'}{K\langle\tilde{a}\rangle \overset{\mu}{\rightsquigarrow} P'} \text{ if } K \overset{\text{def}}{=} (\tilde{b})P$$

$$\texttt{sum:}\ \frac{P \overset{\mu}{\rightsquigarrow} P'}{P + Q \overset{\mu}{\rightsquigarrow} P'} \qquad\qquad \texttt{par:}\ \frac{P \overset{\mu}{\rightsquigarrow} P'}{P \mid Q \overset{\mu}{\rightsquigarrow} P' \mid Q} \text{ if } \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$$

$$\texttt{com:}\ \frac{P \overset{M\,,\,\overline{a}y}{\rightsquigarrow} P' \qquad Q \overset{N\,,\,b(x)}{\rightsquigarrow} Q'}{P \mid Q \overset{L\,,\,\tau}{\rightsquigarrow} P' \mid Q'\{y/x\}} \text{ where } L \overset{\text{def}}{=} \begin{cases} MN[a=b] & \text{if } a \neq b \\ MN & \text{otherwise} \end{cases}$$

$$\texttt{close:}\ \frac{P \overset{M\,,\,\overline{a}(x)}{\rightsquigarrow} P' \qquad Q \overset{N\,,\,b(x)}{\rightsquigarrow} Q'}{P \mid Q \overset{L\,,\,\tau}{\rightsquigarrow} \boldsymbol{\nu}x\,(P' \mid Q')} \text{ where } L \overset{\text{def}}{=} \begin{cases} MN[a=b] & \text{if } a \neq b \\ MN & \text{otherwise} \end{cases}$$

$$\texttt{match:}\ \frac{P \overset{M\,,\,\alpha}{\rightsquigarrow} P'}{[a=b]P \overset{N\,,\,\alpha}{\rightsquigarrow} P'} \text{ where } N \overset{\text{def}}{=} \begin{cases} M[a=b] & \text{if } a \neq b \text{ and } a, b \notin \mathrm{bn}(\alpha) \\ M & \text{if } a = b \end{cases}$$

$$\texttt{res:}\ \frac{P \overset{\mu}{\rightsquigarrow} P'}{\boldsymbol{\nu}b\, P \overset{\mu}{\rightsquigarrow} \boldsymbol{\nu}b\, P'} \text{ if } b \notin \mathrm{n}(\mu) \quad \texttt{open:}\ \frac{P \overset{M\,,\,\overline{a}b}{\rightsquigarrow} P'}{\boldsymbol{\nu}b\, P \overset{M\,,\,\overline{a}(b)}{\rightsquigarrow} P'} \text{ if } b \notin \mathrm{n}(M) \cup \{a\}$$

Table 1. The transition system specialised for open bisimulation

It will be convenient to use a transition system $P \xrightarrow{M\,,\,\alpha} P'$ in which M is in canonical form and substitution σ_M has already been applied to action α and derivative P':

$$\frac{P \overset{N\,,\,\alpha}{\rightsquigarrow} P' \qquad M \text{ is the canonical form for } N}{P \xrightarrow{M\,,\,\alpha\sigma_M} P'\sigma_M} \mathrm{bn}(\alpha) \cap \mathrm{fn}(P) = \emptyset$$

Other notations. If S is a set, then $\wp(S)$ is the powerset of S. If $\mathcal{R}$ is a relation, then we sometimes write $h \mathrel{\mathcal{R}} k$ to mean $(h, k) \in \mathcal{R}$.

Definition 1 (open bisimulation [11]). *Open bisimulation* is the largest set $\{\sim_D\}_D$ of symmetric process relations s.t. for all D and $P \sim_D Q$:

- whenever $P \xrightarrow{M\,,\,\alpha} P'$, with $\mathrm{bn}(\alpha) \cap \big(\mathrm{fn}(Q) \cup \mathrm{n}(D)\big) = \emptyset$ and M respects D, there are N, β, and Q' such that $Q \xrightarrow{N\,,\,\beta} Q'$ and
  - $M \rhd N$,
  - $\alpha = \beta\sigma_M$, and
  - $P' \sim_{D'} Q'\sigma_M$, for $D' \overset{\text{def}}{=} D\sigma_M \cup (\mathrm{en}(\alpha) \times \mathrm{fn}(P, Q))$.

3 Making distinctions local to processes

The goal of this section is to make the indexing distinctions of open bisimulation local to processes. A *constrained process* is a pair $\langle P, D\rangle$, where P is a process and D is a distinction with $\mathrm{n}(D) \subseteq \mathrm{fn}(P)$. The requirement on names is used to keep distinctions "small" and to reduce the number of free names (in the algorithm of Section 6, it will allow us a better re-use of names). The set $\mathcal{CP}$ of constrained processes is ranged over by A, B. Substitutions, free names, bound names are extended to constrained processes as expected. If $A \stackrel{\mathrm{def}}{=} \langle P, D \rangle$, then $\boldsymbol{\nu} a\ A$ abbreviates $\langle \boldsymbol{\nu} a\ P, D - \{a\} \rangle$. We call relations on constrained processes *$\mathcal{CP}$-relations*; they are ranged over by $\mathcal{R}$.

We use $D^{M}_{P,\alpha}$ to denote the distinction $D\sigma_M \cup (\mathrm{en}(\alpha) \times \mathrm{fn}(P))$. Transitions for constrained processes are defined from those for processes:

$$\frac{P \xrightarrow{M\,,\,\alpha} P' \qquad M \text{ respects } D}{\langle P, D \rangle \xrightarrow{M\,,\,\alpha} \langle P', D^{M}_{P,\alpha} \cap \mathrm{fn}(P') \rangle}$$

It makes sense to compare only constrained processes whose distinctions are compatible, that is, identical on common names.

Definition 2. Constrained processes $\langle P, D \rangle$ and $\langle Q, E \rangle$ are *compatible*, written $\langle P, D \rangle \Downarrow \langle Q, E \rangle$, if $D \cap \mathrm{fn}(Q) = E \cap \mathrm{fn}(P)$.

A *compatible* $\mathcal{CP}$-relation is a $\mathcal{CP}$-relation whose pairs are compatible constrained processes.

Theorem 3. *Let $\sim$ be the largest symmetric compatible $\mathcal{CP}$-relation s.t. whenever $\langle P, D \rangle \sim \langle Q, E \rangle$ and $\langle P, D \rangle \xrightarrow{M\,,\,\alpha} \langle P', D' \rangle$, with* $\mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$, *there are N, β, Q' and E' such that $\langle Q, E \rangle \xrightarrow{N\,,\,\beta} \langle Q', E' \rangle$ and*

- *$M \rhd N$,*
- *$\alpha = \beta\sigma_M$, and*
- *$\langle P', D' \rangle \sim \langle Q'\sigma_M, (D^{M}_{P,\alpha} \cup E^{M}_{Q,\alpha}) \cap \mathrm{fn}(Q'\sigma_M) \rangle$.*

Then $\langle P, D \rangle \sim \langle Q, E \rangle$ implies $P \sim_{D \cup E} Q$ and, vice versa, $P \sim_D Q$ implies $\langle P, D \cap \mathrm{fn}(P) \rangle \sim \langle Q, D \cap \mathrm{fn}(Q) \rangle$.

The proof of Theorem 3 uses a few lemmas for manipulating distinctions, among which the following strong-narrowing law for open bisimulation:

$$P \sim_D Q \text{ implies } P \sim_E Q \text{ for } E \stackrel{\mathrm{def}}{=} (D \cap \mathrm{fn}(P)) \cup (D \cap \mathrm{fn}(Q)).$$

This law, which has a delicate proof, strengthens the standard narrowing law for open bisimulation, where $E \stackrel{\mathrm{def}}{=} D \cap \mathrm{fn}(P, Q)$.

A $\mathcal{CP}$-relation $\mathcal{R}$ is *weak transitive* if $A_1\ \mathcal{R}\ A_2$, $A_2\ \mathcal{R}\ A_3$, and $A_1 \Downarrow A_3$ imply $A_1\ \mathcal{R}\ A_3$.

Proposition 4. *Relation $\sim$ is not transitive, but is weak transitive.*

In the remainder of the paper, with some abuse of notation, we call *open bisimilarity* the relation $\sim$ on constrained processes defined in Theorem 3. When clear from the context, we may call constrained processes simply *processes*.

4 Non-redundant transitions

The goal of this section is to remove the dependency called Dep.2 in the Introduction from the characterisation of open bisimulation in the previous section. This dependency is determined by the different conditions in the labels of two matching actions, which may happen when one of the transitions is redundant.

Roughly, a transition $A \xrightarrow{M\,,\,\alpha} A'$ is redundant for open bisimulation if there is another transition $A \xrightarrow{N\,,\,\beta} A''$ which has a strictly weaker condition N and s.t., when M holds, α is equal to β and A' is bisimilar to A''. Redundant transitions can be defined for an arbitrary relation $\mathcal{R}$ in place of bisimilarity.

Definition 5. A transition $\langle P, D\rangle \xrightarrow{M\,,\,\alpha} \langle P', D'\rangle$ is *redundant* for a $\mathcal{CP}$-relation $\mathcal{R}$ if there is a transition $\langle P, D\rangle \xrightarrow{N\,,\,\beta} \langle P'', D''\rangle$ such that:

- $M \not\lhd\!\rhd N$,
- $\alpha = \beta\sigma_M$, and
- $\langle P', D'\rangle \,\mathcal{R}\, \langle P''\sigma_M, D^M_{P,\alpha} \cap \mathrm{fn}(P''\sigma_M)\rangle$.

A transition $\langle P, D\rangle \xrightarrow{M\,,\,\alpha} \langle P', D'\rangle$ is *non-redundant* for a $\mathcal{CP}$-relation $\mathcal{R}$, written $\langle P, D\rangle \xrightarrow{M\,,\,\alpha} \langle P', D'\rangle \in \mathrm{nr}(\mathcal{R})$, if it is not redundant for $\mathcal{R}$.

Lemma 6. *If* $\mathcal{R} \subseteq \mathcal{S}$*, then* $\mathrm{nr}(\mathcal{S}) \subseteq \mathrm{nr}(\mathcal{R})$.

Theorem 7. *Relation* $\sim$ *coincides with the largest symmetric compatible* $\mathcal{CP}$*-relation* $\mathcal{R}$ *s.t. if* $A \,\mathcal{R}\, B$ *then:*

- *whenever* $A \xrightarrow{M\,,\,\alpha} A' \in \mathrm{nr}(\mathcal{R})$ *and* $\mathrm{bn}(\alpha) \cap \mathrm{fn}(B) = \emptyset$*, there is* B' *such that* $B \xrightarrow{M\,,\,\alpha} B' \in \mathrm{nr}(\mathcal{R})$ *and* $A' \,\mathcal{R}\, B'$.

The proof of Theorem 7 exploits the two lemmas below, which express properties of the non-redundant transitions for $\sim$.

Lemma 8. *For each transition* $\langle P, D\rangle \xrightarrow{M\,,\,\alpha} \langle P', D'\rangle$ *there exists a transition* $\langle P', D'\rangle \xrightarrow{N\,,\,\beta} \langle P'', D''\rangle \in \mathrm{nr}(\sim)$ *such that* $M \rhd N$, $\alpha = \beta\sigma_M$ *and* $\langle P', D'\rangle \sim \langle P''\sigma_M, D^M_{P,\alpha} \cap \mathrm{fn}(P''\sigma_M)\rangle$.

Lemma 9. *Suppose* $A \sim B$*. Then for each* $A \xrightarrow{M\,,\,\alpha} A' \in \mathrm{nr}(\sim)$ *with* $\mathrm{bn}(\alpha) \cap \mathrm{fn}(B) = \emptyset$ *there is* B' *such that* $B \xrightarrow{M\,,\,\alpha} B' \in \mathrm{nr}(\sim)$ *and* $A' \sim B'$.

The following lemma is used is Section 5, and compares bisimilar processes on transitions which are non-redundant for a relation larger than $\sim$.

Lemma 10. *Let* $\mathcal{R}$ *be a weak transitive* $\mathcal{CP}$*-relation with* $\sim \subseteq \mathcal{R}$*. If* $A \sim B$ *then for all* $A \xrightarrow{M\,,\,\alpha} A' \in \mathrm{nr}(\mathcal{R})$ *with* $\mathrm{bn}(\alpha) \cap \mathrm{fn}(B) = \emptyset$ *there is some* $B \xrightarrow{M\,,\,\alpha} B' \in \mathrm{nr}(\mathcal{R})$ *such that* $A' \sim B'$.

The proofs of Theorem 7 and of Lemmas 8-10 exploit some powerful proof techniques for $\sim$, which allow us, for instance, to work up to the closure of $\mathcal{CP}$-relations w.r.t. weak transitivity and name substitution when proving bisimilarity results.

5 Active names and the iterative approach

We now will make the choice of bound names of matching transitions local to processes (i.e., removing the dependency called Dep.1 in the Introduction). A local choice cannot be based on the free names because bisimilar processes may have different sets of free names. In place of free names, we shall use the *active names*, which are the same in bisimilar processes. The active names of a process are the smallest subset of free names which affect its behaviour. For instance, a is active in $a(b).\,\mathbf{0}$ and $[a=b]\overline{b}c.\,\mathbf{0}$, but it is not in $\boldsymbol{\nu} b\,(\overline{b}a.\,\mathbf{0})$ and $[a=b]\overline{a}c.\,\mathbf{0}+\overline{b}c.\,\mathbf{0}$. As we did for non-redundant transitions, we first define active names on a generic $\mathcal{CP}$-relation and then consider the active names w.r.t. open bisimulation as a particular case.

Definition 11. For a $\mathcal{CP}$-relation $\mathcal{R}$, the function $\mathrm{an}_{\mathcal{R}} : \mathcal{CP} \to \wp(\mathcal{N})$, mapping a constrained process onto the set of its *active names w.r.t.* $\mathcal{R}$, is the least fixed-point of the monotone function $\psi : (\mathcal{CP} \to \wp(\mathcal{N})) \to (\mathcal{CP} \to \wp(\mathcal{N}))$ defined thus:

$$\psi(f)(A) = \bigcup_{\{M,\alpha,A' \mid A \xrightarrow{M\,,\,\alpha} A' \in \mathrm{nr}(\mathcal{R})\}} \mathrm{fn}(M,\alpha) \cup (f(A') - \mathrm{bn}(\alpha)).$$

If $x \in \mathrm{an}_{\mathcal{R}}(A)$, then we say that name x is *active* w.r.t. $\mathcal{R}$ in process A; otherwise we say that x is *inactive* w.r.t. $\mathcal{R}$ in A.

That is, $\mathrm{an}_{\mathcal{R}}$ is the least function which satisfies the condition:

> whenever $A \xrightarrow{M\,,\,\alpha} A' \in \mathrm{nr}(\mathcal{R})$, it holds that $\mathrm{fn}(M,\alpha) \cup (\mathrm{an}_{\mathcal{R}}(A') - \mathrm{bn}(\alpha)) \subseteq \mathrm{an}_{\mathcal{R}}(A)$.

Lemma 12. *If* $A \sim B$ *then* $\mathrm{an}_{\sim}(A) = \mathrm{an}_{\sim}(B)$.

Proposition 13. $x \in \mathrm{an}_{\sim}(A)$ *iff* $A \not\sim \boldsymbol{\nu} x\, A$.

The following lemma shows that bisimilar processes have the same active names w.r.t. relations larger than $\sim$, under some mild conditions.

Lemma 14. *Suppose that a* $\mathcal{CP}$*-relation* $\mathcal{R}$ *is weak-transitive and closed under injective name substitutions, and that* $\sim\, \subseteq \mathcal{R}$*. Then* $A \sim B$ *implies* $\mathrm{an}_{\mathcal{R}}(A) = \mathrm{an}_{\mathcal{R}}(B)$.

In the *normalised* transitions below, the bound name in a transition of a process A is imposed to be the first inactive name in A; since the first inactive name may also occur free, in rule `Norm2` its free occurrences are redenominated to avoid accidental identifications.

Definition 15. The *normalised transitions for a $\mathcal{CP}$-relation $\mathcal{R}$*, which are of the form $A \xmapsto{M\,,\,\alpha}_{\mathcal{R}} A'$, are defined from the two following inference rules, where $v \overset{\text{def}}{=} \min\{\mathcal{N} - \text{an}_{\mathcal{R}}(A)\}$ and $y \overset{\text{def}}{=} \min\{\mathcal{N} - \text{fn}(A)\}$:

$$\texttt{Norm1}\ \frac{A \xrightarrow{M\,,\,\alpha} A' \qquad \text{bn}(\alpha) \subseteq \{v\}}{A \xmapsto{M\,,\,\alpha}_{\mathcal{R}} A'}$$

$$\texttt{Norm2}\ \frac{A \xrightarrow{M\,,\,\alpha} A' \qquad \text{bn}(\alpha) = \{y\} \qquad y \neq v}{A \xmapsto{M\,,\,\beta}_{\mathcal{R}} A'\{{}^{v}\!/_{y}\ {}^{y}\!/_{v}\}} \text{ where } \beta \overset{\text{def}}{=} \begin{cases} a(v) & \text{if } \alpha = a(y) \\ \overline{a}(v) & \text{if } \alpha = \overline{a}(y) \end{cases}$$

The two inference rules `Norm1-2` define an injective mapping from normalised transitions to plain transitions, because each normalised transition is inferred from a unique plain transition. Thus, if $A \xrightarrow{M\,,\,\alpha} A'$ is the image of $A \xmapsto{M\,,\,\beta}_{\mathcal{R}} A''$ under the injection, we write $A \xmapsto{M\,,\,\beta}_{\mathcal{R}} A'' \in \text{nr}(\mathcal{R})$ if $A \xrightarrow{M\,,\,\alpha} A' \in \text{nr}(\mathcal{R})$.

Definition 16. Let $\mathcal{R}$ be a $\mathcal{CP}$-relation. Function $\Psi_{\mathcal{R}} : \wp(\mathcal{CP} \times \mathcal{CP}) \to \wp(\mathcal{CP} \times \mathcal{CP})$ is defined thus: $(A, B) \in \Psi_{\mathcal{R}}(\mathcal{S})$ if

- $A \mathrel{\mathcal{R}} B$;
- $\text{an}_{\mathcal{R}}(A) = \text{an}_{\mathcal{R}}(B)$;
- whenever $A \xmapsto{M\,,\,\alpha}_{\mathcal{R}} A' \in \text{nr}(\mathcal{R})$, there is B' such that $B \xmapsto{M\,,\,\alpha}_{\mathcal{R}} B' \in \text{nr}(\mathcal{R})$ and $A' \mathrel{\mathcal{S}} B'$;
- the converse of the above clause, with the role of A and B exchanged.

For each $\mathcal{R}$, function $\Psi_{\mathcal{R}}$ is monotone, so it has a greatest fixed-point and we can define a function $\Phi : \wp(\mathcal{CP} \times \mathcal{CP}) \to \wp(\mathcal{CP} \times \mathcal{CP})$, which maps a relation $\mathcal{R}$ onto the greatest-fixed-point of $\Psi_{\mathcal{R}}$.

Functional Φ appears suited to extracting a partition refinement algorithm because derivatives and transitions of processes are computed locally. Notice that in the definition of Φ, clause $A \Downarrow B$ (which appears in the definition of efficient open bisimulation) has been omitted because relation $\Downarrow$ is not transitive, as witnessed by processes

$$A \overset{\text{def}}{=} \langle \boldsymbol{\nu} x\, (\overline{x}a.\,\overline{a}b.\,\mathbf{0}), \emptyset \rangle \qquad B \overset{\text{def}}{=} \langle \mathbf{0}, \emptyset \rangle \qquad C \overset{\text{def}}{=} \langle \boldsymbol{\nu} x\, (\overline{x}a.\,\overline{a}b.\,\mathbf{0}), (a, b) \rangle$$

where $A \Downarrow B$ and $B \Downarrow C$, but $A \Downarrow C$ does not hold. If we added clause $A \Downarrow B$, then function Φ would not preserve transitivity, and hence we could not use Φ to define a partition refinement algorithm.

Unfortunately, Φ is not monotone. Therefore, we do not know whether Φ has a maximal fixed-point and, even if it existed, we could not use Tarsky's theorem to compute it. As a consequence, to obtain an algorithmic characterisation of open bisimulation in terms of Φ, one has to provide a specific proof that the iterated applications of Φ on the universal relation $\mathcal{CP} \times \mathcal{CP}$ are convergent, and that the limit contains open bisimulation. We define:

$$\begin{aligned}
&\Phi^0 \overset{\text{def}}{=} \mathcal{CP} \times \mathcal{CP} \\
&\Phi^i \overset{\text{def}}{=} \Phi(\Phi^{i-1}) \text{ if } i \text{ is an ordinal successor} \\
&\Phi^i \overset{\text{def}}{=} \textstyle\bigcap_{j \le i} \Phi^j \text{ if } i \text{ is an ordinal limit.}
\end{aligned}$$

Lemma 17. *Let $\mathcal{R}$ be weak transitive, closed for injective substitution, and such that $A \mathrel{\mathcal{R}} A\{y/v\}$ whenever $v \notin \mathrm{an}_{\mathcal{R}}(A)$ and $y \notin \mathrm{fn}(A)$. Then $\sim \subseteq \mathcal{R}$ implies $\sim \subseteq \Phi(\mathcal{R}) \subseteq \mathcal{R}$.*

The proof of Lemma 17 depends on Lemmas 10 and 14.

Theorem 18. *Let $\sim_{\mathrm{alg}} \overset{\mathrm{def}}{=} \lim_i \Phi^i$. Then $\sim$ coincides with $\sim_{\mathrm{alg}} \cap \Downarrow$.*

Corollary 19. *It holds that $P \sim_D Q$ if and only if $\langle P, D \cap \mathrm{fn}(P)\rangle \sim_{\mathrm{alg}} \langle Q, D \cap \mathrm{fn}(Q)\rangle$.*

Each iteration of the algorithm in the next section precisely corresponds to an application of function Φ.

6 The algorithm

The algorithm we propose for open bisimulation is based on the characterisation in Corollary 19. The main steps are sketched in Table 2. A partition on processes can be viewed as a relation in which all processes in the same class are related; in this way, we can talk of non-redundant transitions, active names and normalised transitions w.r.t. a partition. Some comments on the steps in the table:

1 Generate the saturated state graphs (S_P, T_P) and (S_Q, T_Q) for processes $\langle P, D \cap \mathrm{fn}(P)\rangle$ and $\langle Q, D \cap \mathrm{fn}(Q)\rangle$.
2 Initialize $\mathcal{P}$ to be the singleton partition on $S_P \cup S_Q$ (i.e., all processes in the same class).
3 Repeat the following steps until partition $\mathcal{P}$ becomes stable:
    3.1 Set *NonRed* to be the subset of transitions in $T_P \cup T_Q$ which are non-redundant for $\mathcal{P}$.
    3.2 Compute the active names w.r.t. $\mathcal{P}$, for each process in $S_P \cup S_Q$.
    3.3 If necessary, refine the partition $\mathcal{P}$ so that processes in the same class have the same set of active names.
    3.4 Compute the normalised transitions for $\mathcal{P}$ generated by the transitions in *NonRed*.
    3.5 Apply the Paige and Tarjan refinement algorithm [10] on partition $\mathcal{P}$ using, as transitions, the normalised ones computed in Step 3.4. Redefine $\mathcal{P}$ to be the resulting partition.
4 Check if $\langle P, D \cap \mathrm{fn}(P)\rangle$ and $\langle Q, D \cap \mathrm{fn}(Q)\rangle$ are in the same class.

Table 2. Schema of the algorithm to check $P \sim_D Q$

Step 1: The *saturated state graph* of a process A_0 is the pair (S, T) where S and T are, respectively, the minimal set of constrained processes and of transitions between processes in S such that $A_0 \in S$ and such that S and T are closed under the following operations:

Sat-trans if $A \in S$ and $A \xrightarrow{M\,,\,\alpha} A'$ with $\mathrm{bn}(\alpha) \subseteq \{\min\{\mathcal{N} - \mathrm{fn}(A)\}\}$, then $A' \in S$ and $A \xrightarrow{M\,,\,\alpha} A' \in T$;

Sat-nonred if $\langle P, D\rangle \xrightarrow{M\,,\,\alpha} \langle P', D'\rangle \in T$ and $\langle P, D\rangle \xrightarrow{N\,,\,\beta} \langle P'', D''\rangle \in T$ with $M \rhd N$ and $\alpha = \beta\sigma_M$, then $\langle P''\sigma_M, D^M_{P,\alpha} \cap \mathrm{fn}(P''\sigma_M)\rangle \in S$;

Sat-bunch Let $y \stackrel{\mathrm{def}}{=} \min\{\mathcal{N} - \mathrm{fn}(A)\}$; if $A \xrightarrow{M\,,\,\alpha} A' \in T$ with $\mathrm{bn}(\alpha) = \{y\}$, then $A'\{{}^{v}\!/\!_{y}\,{}^{y}\!/\!_{v}\} \in S$, for all $v < y$ (where $<$ is the strict order assumed on $\mathcal{N}$).

Sat-nonred and **sat-bunch** are necessary, respectively, for the run-time computation of non-redundant transitions and for the targets of normalised transitions. Note that **sat-trans** requires a single instantiation of bound names of actions. In **sat-bunch**, v is *strictly* below $\min\{\mathcal{N} - \mathrm{fn}(A)\}$, since the latter name, which is surely inactive for A, has already been considered in **sat-trans**.

Step 3: Each cycle corresponds to an application of function Φ of Section 5.

Step 3.1: Following Definition 5, to compute the non-redundant transitions quickly, for each transition $\langle P, D\rangle \xrightarrow{M\,,\,\alpha} \langle P', D'\rangle$ we can keep a list of the processes $\langle P''\sigma_M, D^M_{P,\alpha} \cap \mathrm{fn}(P''\sigma_M)\rangle$ such that $\langle P, D\rangle \xrightarrow{N\,,\,\beta} \langle P'', D''\rangle$ with $M \rhd N$ and $\alpha = \beta\sigma_M$; then the given transition is non-redundant for $\mathcal{P}$ if and only if none of the processes in the list is in the equivalence class of $\langle P', D'\rangle$.

Step 3.2: Following Definition 11, the active names can be efficiently computed via a transitive-closure algorithm (the non-redundant transitions have already been computed in Step 3.1).

Step 3.4: The normalised transitions are generated applying inference rules `Norm1` and `Norm2` (Definition 15) to each transition in *NonRed.* (The derivatives of normalised transitions are in $S_P \cup S_Q$, so no new process needs to be added.)

The algorithm terminates if the saturated state spaces, produced from the input processes in Step 1, are finite. This is the case for finite-control processes.[1].

The algorithm can be used to produce the minimal realisation of a process P w.r.t. a distinction D — for this case it suffices to generate only the saturated state space of $\langle P, D \cap \mathrm{fn}(P)\rangle$ in Step 1. If we take processes with normalised transitions only (normalised transitions for processes are defined as for constrained processes — just replace A with P and A' with P' in Definition 15), then the process returned by the algorithm (i.e., the one extracted from the final partition, where transitions are those computed in the last execution of Step 3.4) is minimal in the number of states, normalised transitions, and the number of free names among all processes in the relation $\sim_D$ with P.

---

[1] Some garbage collection of restrictions is needed, i.e., if $x \notin \mathrm{fn}(P)$ then $\boldsymbol{\nu} x\, P$ should be replaced by P.

7 Conclusions

We have presented various characterisations of open bisimulation. From the last one, we have extracted a partition refinement algorithm. It can be used on finite control processes for checking bisimilarity and for producing minimal realisations of processes.

Complexity. Paige-Tarjan algorithm has a worst-case running time $O(m \log n)$, on the number m of transitions and n of states; in our case, m and n are the number of transitions and of states obtained after initialisation (Step 1 in Table 2). The Paige-Tarjan routine may be applied several times (Step 3 in Table 2), but at most it runs n times, since at each iteration at least one split of the partition is made. Hence, we get a worst case running time $O(mn \log n)$.

However, the algorithm is exponential w.r.t. the syntactic length of the processes. The exponentiality is caused by the expansion of the parallel component of processes, as in CCS, and by value communication, as in data-dependent programs [5]. Surprisingly, the degree of exponentiation in the π-calculus is similar to that of CCS [3, 8]. Against this exponential bounds, the possibility of minimising process representations, offered by the algorithm, becomes important: As it happens in CCS, large-scale examples become tractable if process subcomponents are first minimised.

Active names and non-redundant transitions. The algorithm presented makes no attempts at uncovering active names and non-redundant transitions in initialisation. This may cause a large addition of states in the saturation procedure (operations sat-nonred and sat-bunch in Step 1) and a high number of iterations of the Paige-Tarjan routine.

However, a quick analysis of the processes could often produce reasonable estimates of active names and non-redundant transitions. For instance, all transitions with a true (i.e., empty) condition are non-redundant, and all free names in their labels are active. We believe that this direction can lead to significant improvements.

In non-trivial processes, like those used in the specification and the implementation of the handover protocol in the GSM Public Land Mobile Network [9], all free names are active, and all transitions are non-redundant. It would be useful to find syntactic characterisations of classes of processes with this property. An example is the fragment of language without parallel composition and matching. One could then envisage a two-speed algorithm, the first speed (faster) to be used when active names and non-redundant transitions can be quickly computed in initialisation. (Indeed, the speeds could even be three, the intermediate one to be used when active names are known but non-redundant transitions are not.) Again, the applicability of the first speed can be enhanced by first minimising the representation of process subcomponents whose active names and non-redundant transitions are hard to compute.

Extensions. The algorithm can be extended to the polyadic π-calculus (where tuples of names are communicated). It can also be adapted to early and late bisimulations, and to weak and branching bisimulations. In the case of early and late bisimulations, the algorithm would become simpler, since Dep.2 and Dep.3

are absent, but it would be less efficient, due to the heavy use of names in the input clause of these bisimulations [7, 11].

Implementations. We plan to produce an implementation of the algorithm and to study its integration with the MWB and other tool sets like JACK [1] or AUTO/GRAPH [2].

Minimisation. The algorithm performs minimisation of processes on the normalised transition system. Roughly, normalised means that the bound name of transitions is forced to be the first inactive in the source process.

It would be interesting to see whether there are minimal forms and minimisation algorithms with more relaxed conditions on the choice of bound names. The challenge is that the minimal forms must be canonical, i.e., syntactically identical for bisimilar processes.

Acknowledgements. We are very grateful to Ugo Montanari, for several helpful conversations especially in the initial stages of development of this work. We also benefited from comments by Amar Bouali, Robert de Simone and the anonymous referees.

References

1. A. Bouali, S. Gnesi and S. Larosa. The Integration Project for the JACK Environment. *Bullettin of the EATCS* 54, 1994.
2. G. Boudol, V. Roy, R. de Simone and D. Vergamini. Process algebra and Systems of Communicating Processes. In *Proc. Automatic Verification Methods for Finite State Systems*, LNCS 407, 1989.
3. M. Dam. On the decidability of process equivalences for the π-calculus. SICS Research Report RR:94-20, 1994.
4. J.-C. Fernandez and L. Mounier. “On-the-fly” verification of behavioural equivalences and preorders. In *Proc. CAV'91*, LNCS 575. Springer Verlag, 1994.
5. B. Jonsson and J. Parrow. Deciding bisimulation equivalences for a class of non-finite-state programs. *Information and Computation*, 107:272–302, 1993.
6. P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.
7. R. Milner, J. Parrow and D. Walker. A calculus of mobile processes (parts I and II). *Information and Computation*, 100:1–77, 1992.
8. U. Montanari and M. Pistore. Checking bisimilarity for finitary π-calculus. In *Proc. CONCUR'95*, LNCS 962. Springer Verlag, 1995.
9. F. Orava and J. Parrow. An algebraic verification of a mobile network. *Formal Aspects of Computing*, 4:497–543, 1992.
10. R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
11. D. Sangiorgi. A theory of bisimulation for π-calculus. In *Proc. CONCUR'93*, LNCS 715. Springer Verlag, 1993.
12. B. Victor and F. Moller. The Mobility Workbench — A tool for the π-calculus. In *Proc. CAV'94*, LNCS 818. Springer Verlag, 1994.