Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 130 (2005) 169–185
www.elsevier.com/locate/entcs

From Active Names to π-*calculus* Rewriting Rules

Ana C. V. de Melo

University of São Paulo (USP)
Department of Computer Science
Rua do Matão, 1010 - Cidade Universitária
05508 090 - São Paulo - SP - Brazil

Abstract

The problem of checking equivalences for π-agents is not trivial and has been widely studied in the last decade. Syntactic and semantic approaches can be taken to formally verify π-calculus equivalences. The syntactic approach rests mainly on structural congruence. On the other hand, the semantic checking methods can verify wider equivalences but cannot check infinitary π-agents. Bisimilar agents have the same set of active names. This result and a technique to check bisimulation considering active names is presented in [7]. There, agents active names are calculated from their corresponding Labelled Transition Systems (LTS) and, because of this, cannot be directly applied to rewriting systems. In [2], a syntactic characterisation of active names for π-agents was presented. Here, new rewriting rules are presented (based on the syntactic characterisation of active names) to identify and discard useless code of π-expressions for a class of expressions including composition. With these new rules, π-expressions are better reduced (more useless code is discarded) enriching the equivalence classes of agents.

Keywords: mobile agents, π-*calculus*, formal verification, rewriting systems

1 Introduction

The problem of formal verification for π-*calculus* agents has been widely studied in the last decade. Regarding equivalences checking, we can roughly split verification techniques into those that address the problem at the syntactic level using rewriting systems, for example [3]; and those based on labelled transition systems (LTS) for the behavioural equivalences. In both cases, the problem of checking equivalences for π-agents is not trivial and each technique has limitations.

doi:10.1016/j.entcs.2005.03.010

For checking behavioural equivalences, state explosion is a problem due to input actions raising an infinite number of transitions when the set of agents names is infinite. Montanari and Pistore [7,8] proved that finite automata can be efficiently built for finitary [1] π-*calculus* without matching (representative inputs are selected). From those finite branching automata, **active names** are calculated and the automata are further reduced to check equivalence (bisimilar π-agents have the same set of active names). For this approach, calculation of agents active names depends on labelled transition systems (automata are built) and is limited to the class of finitary π-*calculus* without matching.

For the syntactic approach, π-expressions are rewritten into normal forms and agents are equivalent if their expressions are normalised to the same expression modulo to alpha conversion, commutativity and associativity of parallel composition and permutation of consecutive restrictions. Most verification systems in this approach are concerned with structural congruence. Some efforts have been employed to improve syntactic verification techniques to check equivalences wider than structural congruence. Hirschkoff [3,4], for example, developed a technique to extend the idea of checking structural congruence to bisimulation-up-to by Sangiorgi [11,12].

In [2], a study on the syntactic characterisation of potential active names was presented. That study identified certain classes of expressions for which active names can be syntactically calculated and others for which internal communications are required to calculate active names. Despite of giving directions on how to calculate active names and its counterpart inactive names, the study does not give insights on the use of the latter to eliminate useless code of π-expressions. The elimination of useless code of π-expressions is of interest to slicing techniques and, in particular, to rewriting verification techniques.

This paper presents an application of the active names syntactic characterisation to reduce π-expressions and enhance π-*calculus* rewriting verification techniques. This is achieved with the development of new rewriting rules to eliminate useless code from π-expressions at the normalisation stage. The syntactic calculation of active names is not possible in general and internal communications are necessary for certain classes of expressions. The rewriting rules presented here are restricted to classes of π-expressions for which active names calculation remains decidable.

π-*calculus* preliminaries are first presented: the language fragment and its semantics. Section 3 summarises the study of active names. Section 4 presents

[1] "an agent is finitary if there is a bound to the number of parallel components of all the agents derivable from from it. ...A syntactical but more restrictive notion is that of finite control agents, i.e., the agents without parallel composition inside recursion." [7].

a rewriting system followed by the presentation of new rewriting rules. The last section analyses the limitations of the present work.

2 π-*calculus* Preliminaries

The π-*calculus* fragment (monadic π-*calculus*) used in the present work is similar to the one in [6], but replicated agents are guarded by a prefixing action [2]:

$$Q ::= \mathsf{0} \mid \alpha.P \mid (\nu\vec{x})\ P \mid P_1 + P_2 \mid P_1 \mid P_2 \mid !\ \alpha.P$$

The language elements have the usual meaning. $\mathsf{0}$ represents the stop agent and cannot perform actions. $(\nu\vec{x})\ P$ makes all elements in $\vec{x}$ restricted to agent P. $P_1 + P_2$ represents the choice of either agent P_1 or P_2, while $P_1 \mid P_2$ represents composition of agents. Finally, $!\ \alpha.P$ replicates $\alpha.P$ as much as required; there is an infinite number of $\alpha.P$s in composition.

Agent actions α are defined as follows:

$$\alpha ::= \tau \mid a(b) \mid \overline{a}b \mid \overline{a}(b)$$

τ is the silent (internal) action: an action with no observable behaviour. $a(b)$ denotes an action receiving b along port a, $\overline{a}b$ denotes an action sending b along port a, and $\overline{a}(b)$ sends the internal name b along port a to its context (scope extension). $\mathcal{I}$ and $\mathcal{O}$ denote the sets of input and output actions respectively. $ch(\alpha)$ denotes the action port name while $obj(\alpha)$ represents the information passed along the port. The restriction $(\nu b)\ P$ and the input action $a(b).P$ both bind name b to the scope of P. b is a *bound name* (bn) in both cases, and a *free name* (fn) otherwise.

2.1 Semantics

One way of defining π-*calculus* semantics is first capturing the notion of structural congruence and then define behaviour by transition means. Structural congruence ($\equiv$) is defined as the smallest congruence satisfying laws in Table 1 [3] [9] (laws over replicated agents have been added to the original definition in order to handle the rewriting system by Hirschkoff [3,4]).

Structural congruence is not enough to define behavioural equivalences in process algebras. Besides structural congruence, an operational semantics of

[2] Guarded replicated agents is necessary to find a normal form in rewriting systems. Also, summation is not considered in the rewriting system shown later; it has been introduced here because of the active names study.

[3] There are many ways of defining structural congruence, we have chosen one to help finding rewriting rules.

(i) If P and Q are variants of alpha-conversion then $P \equiv Q$.

(ii) The Abelian monoid laws for Parallel and Sum:
- (a) commutativity: $P \mid Q \equiv Q \mid P$, $P + Q \equiv Q + P$;
- (b) associativity: $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$, $(P + Q) + R \equiv P + (Q + R)$); and
- (c) $\mathsf{0}$ as unit: $P \mid \mathsf{0} \equiv P$ and $P + \mathsf{0} \equiv P$.

(iii) The scope extension laws:
- (a) $(\nu x)\ \mathsf{0} \equiv \mathsf{0}$
- (b) $(\nu x)\ P \equiv P$, if $x \notin \mathsf{fn}(P)$
- (c) $(\nu x)\ (P \mid Q) \equiv (P \mid (\nu x)\ Q)$, if $x \notin \mathsf{fn}(P)$
- (d) $(\nu x)\ (P + Q) \equiv (P + (\nu x)\ Q)$, if $x \notin \mathsf{fn}(P)$
- (e) $(\nu x)\ (\nu y)\ P \equiv (\nu y)\ (\nu x)\ P$

(iv) The replication laws:
- (a) $!\ \alpha.P \mid \alpha.P \equiv !\ \alpha.P$
- (b) $!\ \alpha.P \mid !\ \alpha.P \equiv !\ \alpha.P$

Table 1
Structural Congruence Rules

each combinator is necessary. Here, the semantics of π-*calculus* is given by a labelled transition system based on late semantics (Table 2) [9].

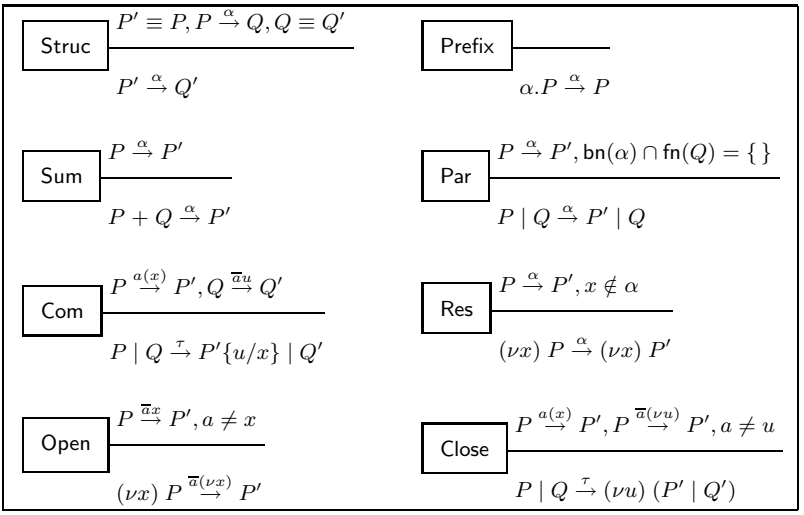

Table 2
Transition Semantics (late) for π-*calculus*

Rule Struc is introduced to take structural congruence into account. This also simplifies the transition system rules. Since commutativity laws are defined for summation and parallel composition in the structural congruence, there is no need to define the dual rules of Sum and Par.

There is a set of equivalences defined for π-*calculus*[9]. Here, only early bisimulation is defined.

Definition 2.1 An early bisimulation ($\sim$) with late semantics is a symmetric binary relation $\underline{R}$ on agents satisfying the following: $P\underline{R}Q$ and $P \xrightarrow{\alpha} P'$, where $\mathsf{bn}(\alpha)$ is fresh, implies that

1. if $\alpha = a(x)$ then $\forall u : \exists Q' : Q \xrightarrow{a(x)} Q' \wedge P'\{u/x\}\underline{R}Q'\{u/x\}$, and;
2. if α is not an input, then $\exists Q' : Q \xrightarrow{\alpha} Q' \wedge P'\underline{R}Q'$.

P and Q are (strongly) early bisimilar, written $P \sim Q$, if they are related by an early bisimulation.

3 Active Names

Active names of π-*calculus* agents (π-agents for short) have been studied by Montanari and Pistore in [7] based on the idea of **used names** for value passing CCS [5]. Pistore and Sangiorgi [10,11,12] proposed a partition refinement algorithm to check early and open bisimulations using active names calculated from LTS. In [2], a study was carried out to recognise active names from π-agents expressions, instead of LTS.

A name is said semantically active in agent P if it is a free name and can be performed by P. An important result from this rests on bisimilar agents having the same set of active names: *"the active names of an agent is the smallest subset of free names which affects the agent behaviour"*. This is formally stated in [7] as follows:

Definition 3.1 A name a is active for an agent P iff $P \not\sim (\nu a)\ P$; $\mathsf{an}(P) = \{a | P \not\sim (\nu a)\ P\}$ is the set of active names for the agent P.

Proposition 3.2 *If* $P \sim P'$ *then* $\mathsf{an}(P) = \mathsf{an}(P')$.

A name is semantically inactive for an agent if it is unable to change such agent from the external context point-of-view. Bound names can play a role on internal actions of agents (they are thus unable to interfere on the external context). Besides that, certain actions are never performed due to names restriction. So, agent names exclusively involved in either internal actions or actions never engaged (useless actions) are the so called inactive names.

Certain π-expressions can have active names identified without engaging into internal actions. Table 3 summarises the syntactic characterisation of active names for that class of expressions.

Example 3.3 $Q1 \stackrel{def}{=} a(x).x(y) \mid \overline{a}b.\overline{b}c.c(z)$

	Agent	Active Names
1	0	$\{\}$
2	$\alpha.P$	$\mathsf{fn}(\alpha) \cup (\mathsf{an}(P) - \mathsf{bn}(\alpha))$
3	$P_1 + P_2$	$\mathsf{an}(P_1) \cup \mathsf{an}(P_2)$
4	$(\nu\vec{x})\ \alpha.P$	$\begin{cases} \{\} & , ch(\alpha) \in \vec{x} \\ \mathsf{fn}(\alpha) \cup (\mathsf{an}((\nu\vec{x} - \{b\})\ P) - \mathsf{bn}(\alpha))) & , \alpha = \overline{a}(b) \wedge a \notin \vec{x} \\ \mathsf{fn}(\alpha) \cup (\mathsf{an}((\nu\vec{x})\ P) - \mathsf{bn}(\alpha)) & , otherwise \end{cases}$
5	$(\nu\vec{x})\ (P_1 + P_2)$	$\mathsf{an}((\nu\vec{x})\ P_1) \cup \mathsf{an}((\nu\vec{x})\ P_2)$
6	$\alpha.P_1 \mid \beta.P_2$	$\mathsf{an}(\alpha.P_1) \cup \mathsf{an}(\beta.P_2)$
7	$!\ \alpha.P$	$\mathsf{an}(\alpha.P)$
8	$!\ \alpha.P_1 \mid !\ \beta.P_2$	$\mathsf{an}(\alpha.P_1) \cup \mathsf{an}(\beta.P_2)$

Table 3
Active Names without Reaction

$$
\begin{aligned}
\mathsf{an}(Q1) &= \mathsf{an}(a(x).x(y)) \cup \mathsf{an}(\overline{a}b.\overline{b}c.c(z)) \\
\mathsf{an}(a(x).x(y)) &= \mathsf{fn}(a(x)) \cup (\mathsf{an}(x(y)) - \mathsf{bn}(a(x))) \\
&= \{a\} \cup (\mathsf{fn}(x(y)) \cup (\{\} - \mathsf{bn}(x(y))) - \{x\}) \\
&= \{a\} \cup (\{x\} \cup (\{\} - \{y\}) - \{x\}) \\
&= \{a\} \cup (\{x\} - \{y\} - \{x\}) \\
&= \{a\} - \{y\} \\
&= \{a\} \\
\mathsf{an}(\overline{a}b.\overline{b}c.c(z)) &= \mathsf{fn}(\overline{a}b) \cup (\mathsf{an}(\overline{b}c.c(z)) - \mathsf{bn}(\overline{a}b)) \\
&= \{a, b\} \cup (\mathsf{fn}(\overline{b}c) \cup (\mathsf{an}(c(z)) - \mathsf{bn}(\overline{b}c) - \{\}) \\
&= \{a, b\} \cup (\{b, c\} \cup (\{c\} \cup (\{\} - \{z\}) - \{\}) - \{\}) \\
&= \{a, b, c\} \\
\mathsf{an}(Q1) &= \{a\} \cup \{a, b, c\} = \{a, b, c\}
\end{aligned}
$$

Compound agents can engage into internal communications and communicate with the external environment using the same actions because there is no restriction on the top of expression. As a result, no scope extrusion is made and all actions in the expression can be engaged. For agents with no restricted names, active names can be calculated directly from expressions.

For the general case, agent active names calculation is not possible without engaging into internal communications. For agents that can exclusively engage into internal actions, internal steps must be performed to check which actions come next and, then, calculate their active names. Table 4 summarises the

calculation of potential agents active names whenever internal communications are required.

In order to get all possible internal communications for replicated processes, we must then consider at most the number of subsequent actions an agent can perform. This corresponds to the size of expressions ($\mathsf{expsize}(E)$ is given by the number of sequent actions, item 3) considering only prefixing and composition operators.

	Agent	**Active Names**
1	$(\nu\vec{x})\ \alpha.P \mid \beta.Q$	$\bigcup \begin{cases} (\mathsf{an}((\nu\vec{x})\ \beta.Q)) \cup (\mathsf{an}((\nu\vec{x})\ \alpha.P)), \\ (\mathsf{an}((\nu\vec{x})\ (P\{c/b\} \mid Q)) - \mathsf{bn}(\alpha)), & \alpha = a(b) \wedge \\ & \beta = \overline{a}c \wedge \\ & a \in \vec{x} \\ (\mathsf{an}((\nu\vec{x} \cup \{c\})\ (P\{c/b\} \mid Q)) \\ \qquad -\mathsf{bn}(\alpha) - \mathsf{bn}(\beta)), & \alpha = a(b) \wedge \\ & \beta = \overline{a}(c) \wedge \\ & a \in \vec{x} \end{cases}$
2	$(\nu\vec{x})\ (\sum_{i=1}^{n} \alpha_i.P_i \mid \sum_{j=n+1}^{m} \alpha_j.Q_j\ )$	$\bigcup\{\mathsf{an}((\nu\vec{x})\ (\alpha_i.P_i \mid \alpha_j.Q_j))\mid 1 \leq i \leq n, n+1 \leq j \leq m\}$
3	$(\nu\vec{x})\ !\ \alpha.P$	$\mathsf{an}((\nu\vec{x})\ \prod_{i=1}^{n} \alpha.P\ ), n = \mathsf{expsize}(\alpha.P) + 1$
4	$(\nu\vec{x})\ \prod_{i=1}^{n} !\ \alpha_i.P_i$	$\bigcup \begin{cases} \bigcup\{\mathsf{an}((\nu\vec{x})\ (!\ \alpha_i.P_i))\mid 1 \leq i \leq n\} \\ \bigcup\{\mathsf{an}((\nu\vec{x})\ (\alpha_i.P_i \mid \alpha_j.P_j))\mid 1 \leq i, j \leq n \wedge i \neq j\} \end{cases}$

Table 4
Active Names under Reaction

The example below shows how to calculate active names as internal communications are required.

Example 3.4 Suppose agents $Q2$ and $Q3$ defined as follows:

$Q2 \stackrel{def}{=} \overline{c}d.a(x).\overline{x}y$

$Q3 \stackrel{def}{=} \overline{a}(b).b(z).\overline{y}w$

$(\nu a)\ Q2 \stackrel{\overline{c}d}{\rightarrow} (\nu a)\ (a(x).\overline{x}y) \sim \mathsf{0}$

$(\nu a)\ Q3 \sim \mathsf{0}$

$(\nu a)\ (Q2 \mid Q3) \stackrel{\overline{c}d}{\rightarrow} (\nu a)\ (a(x).\overline{x}y \mid Q3) \stackrel{\tau}{\rightarrow} (\nu a, b)\ (\overline{b}y \mid b(z).\overline{y}w)$
$\stackrel{\tau}{\rightarrow} (\nu a, b)\ (\mathsf{0} \mid \overline{y}w) \stackrel{\overline{y}w}{\rightarrow} (\nu a, b)\ (\mathsf{0} \mid \mathsf{0})$

$\mathsf{an}((\nu a)\ Q2) = \{c, d\}$

$\mathsf{an}((\nu a)\ Q3) = \{\}$

$\mathsf{an}((\nu a)\ (Q2 \mid Q3)) = \{c, d, y, w\}$

The example above shows a case in which scope extension is performed and then actions can be engaged due to such names extrusion, otherwise the agent would stop. Agent $(\nu a)\ Q2$, for example, is unable to perform action $a(x)$ because name a is restricted. For the same reason $(\nu a)\ Q3$ has a behaviour bisimilar to agent stop. Although both agents are not able to progress in isolation due to restricted names, they can engage into internal communications when they are composed and restriction is put on the top of the whole agent scope. As a result, $(\nu a)\ (Q2 \mid Q3)$ can perform all $Q2$ and $Q3$ actions and the active names set comprises all those action names. Note that scope extrusion of name b is first performed due to the internal action, and this makes the second internal action possible.

4 A Rewriting System for π-*calculus*

Checking bisimulation can be performed by rewriting systems concerned with structural congruence [12]: a normal form for π-expressions is used and agents are bisimilar if they can be re-written to the same expression modulo to alpha conversion, commutativity and associativity of parallel composition and permutation of consecutive restrictions. Besides that, Hirschkoff [4] developed a technique to extend the idea of checking structural congruence to bisimulation-up-to, by Sangiorgi [11].

This section summarises the rewriting system to check bisimulation-up-to developed by Hirschkoff [3,4]. New rewriting rules based on the study of active names are described afterwards (Section 5).

4.1 Checking Structural Congruence through Rewriting

The verification technique developed by Hirschkoff handles a fragment of π-*calculus* without summation, match or mismatch. For that, a normal form was defined and proved unique[3]:

$$(1)\quad P = \mathsf{0} \mid (\nu\vec{x})\ \alpha_1.P_1 \mid \alpha_2.P_2 \mid \ \ldots\ \mid \alpha_m.P_m \mid !\ \alpha_{m+1}.P_{m+1} \mid \ \ldots\ \mid !\ \alpha_n.P_n \mid$$

Note that summation has not been considered in this normal form and all agents are product of prefixed agents. Also, alpha-conversion is used in order to avoid free and bound names clash. This normal form is also considered for the definition of new rewriting rules in Section 5.

To check agents equivalence, a term normalisation algorithm is first applied and, then, normalised expressions are compared. The normalisation is as follows:

Definition 4.1 The normalisation algorithm can be defined as a rewriting system based on the following rules:

[**R1**] $P \mid \mathsf{0} \rightarrow P$

[**R2**] $(\nu x)\ P \rightarrow P$, if $x \notin \mathsf{fn}(P)$

[**R3**] $P \mid (\nu x)\ Q \rightarrow (\nu x)\ (P \mid Q)$, if $x \notin \mathsf{fn}(P)$

[**R4**] $! \ \alpha.P \mid \alpha.P \rightarrow ! \ \alpha.P$

[**R5**] $! \ \alpha.P \mid ! \ \alpha.P \rightarrow ! \ \alpha.P$

[**R6**] $(\nu \vec{x})\ \alpha.P \rightarrow \mathsf{0}$, if $ch(\alpha) \in \vec{x}$

Apart from rule [**R3**], the rewriting system is made of reduction rules so that their application to expressions leads to new expressions shorter in size. The great majority of rules in Definition 4.1 are taken from those of structural congruence (Table 1). As a result, reduced expressions are structural congruent to the original ones and an algorithm can be found to reduce expressions to their corresponding normal forms. Confluence of this system (including rules [**R1**]–[**R5**]) has been proved in [3].

Rules [**R1**] to [**R5**] come all from structural congruence definitions. Rule [**R1**] reduces any agent composed with the stop agent to itself. Rule [**R2**] eliminates the restriction combinator whenever the restricted names do not appear free in the agent [4]. Rule [**R3**] is about scope extension. Differently from the previous rules, its application keeps the expression size instead of reducing it. Nevertheless, all restrictions are pulled up as much as possible and the reduction procedure stops when the rule is no longer applicable.

Rules [**R4**] and [**R5**] are about replicated agents. In [**R4**], an agent is removed if composed with a replicated copy of itself. In [**R5**], a replicated agent is removed when composed with itself. Note that these rules are only possible because all replicated agents are guarded by an action; this is not true for arbitrary replicated agents, as noted by Milner in [6]. All other rules from structural congruence definitions with no ability to reduce expressions are denied.

[**R6**] is not catched from congruence rules. It was further created [4] to eliminate unused parts of expressions and is based on the idea of removing agents that cannot have their prefixing actions engaged. Note that this rule is not very concerned with structural congruence because it has semantical (on behaviour) rather than geometrical meaning. This was, however, introduced as a "congruence" rule to make easier the application of up-to-bisimulation technique.

[4] Rules [**R2**] and [**R3**] are guarded by a condition about name freeness. Even though, the system is considered an ordinary Term Rewriting System since such a condition can be embedded in agents representation if a De Bruijn notation for names is used.

Example 4.2 Consider the following agents:

$Q4 \stackrel{def}{=} a(x).x(w)$

$Q5 \stackrel{def}{=} (\nu y)\ (a(x).x(w).\overline{y}(b).\overline{b}c.c(z))$

Despite having different sets of free names, both agents can only perform the first two actions because y is restricted. To check that these agents are bisimilar, the rewriting system above is first applied to both agents so that $Q5$ is reduced to $a(x).x(w)$:
since

Agent	Reduced Agent	Rule
$(\nu y)\ (\overline{y}(b).\overline{b}c.c(z)) \rightarrow$	0	**[R6]**

the whole agent $Q5$ is reduced to $a(x).x(w)$

Agent	Reduced Agent	Rule
$(\nu y)\ (a(x).x(w).\overline{y}(b).\overline{b}c.c(z)) \rightarrow$	$(\nu y)\ (a(x).x(w))$	**[R6]**
$(\nu y)\ (a(x).x(w)) \rightarrow$	$a(x).x(w)$	**[R2]**

$Q4$ and $Q5$ expressions become identical after reduction.

Besides structural congruence, Hirschkoff also checked bisimulation-up-to injective substitution, restriction and composition. To check up-to-substitution and up-to-restriction, normalised agents are used and an algorithm that work on injective substitutions of free names is applied (it is, in fact, an extension of the alpha-conversion checking method). To check up-to-composition, however, transitions on normalised expressions must be performed: communications resulting in silent actions are reduced. These reductions have two main consequences: first, the original agents are not recovered after internal communications; second, a set of new expressions may raise as internal communications are performed. However, it still controls "state" explosion because only silent actions are performed.

Even with the new rule ([**R6**]), the rewriting system is unable to remove all inactive actions from expressions (bisimilar agents have the same set of active actions [7]). Agents having a prefixing action unable to communicate with its context due to port name restriction has already been captured by rule [**R6**]. Cases other than prefixing actions, however, may also lead to agents unable to communicate with their contexts, as shown in the following example.

Example 4.3 Consider agents $Q6$ and $Q7$:

$Q6 \stackrel{def}{=} (\nu a)\ (a(x).x(w) \mid \overline{b}c)$

$Q7 \stackrel{def}{=} \overline{b}c$

Action $a(x)$ (agent $Q6$) is neither able to communicate with the external environment (a is restricted) nor with the other compound agent ($\overline{b}c$). As a result, $a(x).x(w)$ could be removed from $Q6$ without affecting its behaviour. If so, $Q6$ would become identically written to agent $Q7$ (they are bisimilar).

Bisimilarity of $Q6$ and $Q7$ is not recognised by the rewriting system above: no rule can be applied to rewrite $Q6$ into $Q7$. To extend the idea of finding inactive actions of π-expressions, the following section presents new rules to eliminate inactive actions from π-expressions based on the study of active names. The new set of rules does not cover late or early bisimulations for the fragment of *π-calculus* with replicated agents, but it is a step forward in the sense of slightly enriching terms reduction used in [4] with composition.

5 From Inactive Names to New Rewriting Rules

With the syntactic characterisation of active names we can find both the potential active and inactive names of agents. Inactive names are the union of the bound names set with free names that exclusively appear in actions never engaged by agents (namely inactive actions). Analysing expressions and their corresponding sets of active names, we may find subexpressions never used when preceded by inactive actions. These unengageable subexpressions can be substituted for the stop agent preserving bisimulation. In order to remove unused subexpressions, the study on the characterisation of active names is used as a foundation to reduce π-expressions and then enrich *π-calculus* verification techniques based on term rewriting. This section presents new rewriting rules based on that study.

5.1 Inactive Actions: cases

From syntactic characterisation of active names, we may also find certain actions never engaged due to names restriction. Moreover, actions preceded by one never engaged (inactive action) become inactive. Restricted names are, in fact, the key-point for unengageable actions. This section summarises cases in which restricted names lead to unused subexpressions (useless code) for the *π-calculus* fragment with prefixing and parallel composition.

Inactive names can be split into the set of free and bound names. All inactive free names exclusively appear in inactive actions. On the other hand, the inactive bound names may participate in engageable and unengageable actions. The set of engageable bound names is made of all bound names that appear in the calculation of active names. Bound names not appearing in such a calculation belongs to actions never engaged (inactive actions). One

can identify candidates for unengageable actions from active/inactive names calculation. Here, situations that lead to potential unengageable actions are analysed for expressions involving restricted prefixed and composed agents:

Prefixing: The port name of a prefixing action is restricted – the prefixing action cannot be engaged and the subsequent actions are also inactive (already captured by rule **[R6]** and related to item 4 of Table 3 - first line).

Composition: An agent is a composition of two processes with restricted port names so that one of the processes cannot communicate with the context nor with the other process.

We must first analyse how a restricted action **can communicate** with others in order to find the opposite situations. Let α be a prefixing restricted action of agent P composed with Q ($(\nu\vec{x})\ (\alpha.P \mid Q)$ and $ch(\alpha) \in \vec{x}$). α can communicate with Q in one of the following ways:

(i) there exists action $\overline{\alpha}$ in Q. Since a bound name is different from all other bound or free names in expressions, $\overline{\alpha}$ port name cannot be reconfigured. Whenever $\overline{\alpha}$ is in Q, it can communicate to α (ex.: $(\nu a)\ (a(x).A \mid \overline{a}b.B)$);

(ii) there exists action β in agent Q that extrudes α port name to the context and the β port name is free (β is an engageable action). (ex.: $(\nu a)\ (a(x).A \mid \overline{b}(a).B)$); or

(iii) there exists a third agent inside the restriction. For example,
$(\nu\vec{x})\ (\alpha.P \mid Q \mid R)$ and $ch(\alpha) \in \vec{x}$
and β (in agent Q) can be reconfigured to $\overline{\alpha}$ due to an internal communication with the third agent (R). A particular example:
$(\nu a, b)\ (a(x).A \mid b(y).\overline{y}c.B \mid \overline{b}a.C)$.

If one of these cases is found, the restricted agent is able of engaging into communications. **For the opposite situations**, however, agents **cannot communicate** and the useless subexpressions can be removed. All these potential communications are related to item 1 of Table 4).

The following sections formalise the situations above to denote useless subexpressions throughout rewriting rules.

5.2 Preliminary Definitions

In order to define new rules, we first give basic definitions. The following definitions are all related to agent expressions (syntax) instead of actions semantics. Let $\mathcal{I}$ be the set of input actions ($\beta \in \mathcal{I}$ means β is an input action) and $\mathcal{O}$ the set of output actions.

Definition 5.1 Action equality up to communication and alpha-conversion is defined as: α equals β (denoted as $\alpha \rightleftharpoons \beta$) if

$ch(\alpha) = ch(\beta) \wedge ((\alpha \in \mathcal{I} \wedge \beta \in \mathcal{I}) \vee (\alpha \in \mathcal{O} \wedge \beta \in \mathcal{O}))$

Note that to have the same communication capability, actions must have identical port name and both be input or output actions.

Actions can communicate to each other whenever they have the same port name and one of them is an input and the other is an output action. The following definition formalises the idea of having complementary actions undertaking communication notion.

Definition 5.2 An action α is complementary to β if they can communicate to each other:

$ch(\alpha) = ch(\beta) \wedge ((\alpha \in \mathcal{I} \wedge \beta \in \mathcal{O}) \vee (\alpha \in \mathcal{O} \wedge \beta \in \mathcal{I}))$

Note that an input action may communicate to either output or output bound actions. An action is said to appear, or belong to, an agent expression, if it is written down in the agent expression.

Definition 5.3 An action α is in agent P (denoted as $\alpha \in P$) if it, or its alpha-conversion, appears in P's definition[5]: $P \stackrel{def}{=} \ldots\alpha\ldots$

An action can be reconfigured if it is preceded by an input action and its channel name coincides with the object name of that input action. This notion of action reconfiguration is defined as follows:

Definition 5.4 An action β reconfigures action α in an agent P (denoted as $\beta \hookrightarrow \alpha : P$) if:

$P \stackrel{def}{=} \ldots\beta.P_1.\alpha\ldots \wedge \beta \in \mathcal{I} \wedge ch(\alpha) = obj(\beta)$

Once an action can be reconfigured, it can evolve to a particular action. It can, indeed, have its names changed to become identical to another action up to alpha-conversion.

Definition 5.5 Action β can evolve to α in agent $(\nu\vec{x})\ P$ (denoted as $\beta \searrow \alpha : (\nu\vec{x})\ P$) if

$\beta \in (\nu\vec{x})\ P \wedge$

$(\beta = \alpha \wedge \nexists\gamma.\gamma \hookrightarrow \beta : (\nu\vec{x}P)\ ) \vee$

$(\exists\gamma.\gamma \hookrightarrow \beta : (\nu\vec{x}P) \wedge \beta\{ch(\alpha)/ch(\beta)\} \rightleftharpoons \alpha)$

Action β becomes equal (up to communication) to α if its port name is renamed to α port name. Note that this equality is in the sense of Definition 5.1 – concerned with communication ability.

[5] This is an abuse of notation to make further definitions easier.

5.3 New Rules

Here, new rewriting rules are created to reduce π-expressions as agents are unable to engage into certain actions.

A priori, we could think of reducing agents expressions whenever an action to be performed has restricted port name. This might be true if we had neither communication through composition nor reconfiguration and extruded names in π-agents, as shown in the active names study. In [4], Hirschkoff developed a rewriting system based on structural congruence enriched with a rule to reduce agents expressions as restrictions are on prefixing actions port names (rule [**R6**]). That rule, however, does not cover all situations in which subexpressions can be removed without affecting agent behaviour (already discussed in Section 4.1).

Here, rules to reduce agents expressions as a restricted action is found and the agent cannot communicate with its context are defined (situations shown in Section 5.1 - **Composition**). In fact, this extends the rewriting system to also handle useless subexpressions in composed agents.

Definition 5.6 The rewrite rules for parallel agents are defined as follows:

$$[\mathbf{R7}]\ (\nu\vec{x})\ (\alpha.P \mid Q) \rightarrow (\nu\vec{x})\ Q \qquad ,if ch(\alpha) \in \vec{x} \wedge \nexists \overline{\alpha} \in Q \wedge \nexists \beta \in \mathcal{O}.\beta \in Q \wedge obj(\beta) = ch(\alpha)$$

$$[\mathbf{R8}]\ (\nu\vec{x})\ (\alpha.P \mid Q \mid R) \rightarrow (\nu\vec{x})\ (Q \mid R) \ ,if(ch(\alpha) \in \vec{x} \wedge \exists \beta \in \mathcal{O}.(\beta \in R \wedge obj(\beta) = ch(\alpha) \wedge ch(\beta) \in \vec{x}) \wedge \nexists \gamma \in Q.\gamma \searrow \overline{\alpha} : Q)$$

For rule [**R7**], agent $\alpha.P$ is reduced to $\mathsf{0}$ because it cannot communicate to Q; Q has no $\overline{\alpha}$ action. At the same time, Q is not able to extrude the α port name to the context. This rule formalises items **Composition** 1 and 2 from Section 5.1.

Example 5.7 Consider agents $Q6$ and $Q7$ from **Example 4.3**:

$Q6 \stackrel{def}{=} (\nu a)\ (a(x).x(w) \mid \overline{b}c)$

$Q7 \stackrel{def}{=} \overline{b}c$

Applying Rule [**R7**] to agent $Q6$, we obtain a new agent that can be further

reduced to agent $Q7$:

Agent		Reduced Agent	Rule
$(\nu a)\ (a(x).x(w) \mid \overline{b}c)$	$\rightarrow$	$(\nu a)\ \overline{b}c$	**[R7]**
$(\nu a)\ \overline{b}c$	$\rightarrow$	$\overline{b}c$	**[R2]**

$Q6$ and $Q7$ are checked bisimilar as normalisation is applied.

On the other hand, rule **[R8]** shows how to reduce an agent expression even when the α port name is apparently extruded but no action in Q can evolve to $\overline{\alpha}$. At a first glance, this looks inconsistent with rule **[R7]** in which the α channel name is extruded. However, if action that extrudes the channel name is also restricted, the only way to have a communication with α is when it is given as input to Q (from R), and a Q's action can evolve to complementary α ($\overline{\alpha}$). Rule **[R8]** gives a syntactic situation in which such a communication is impossible and formalises item **Composition** 3 of Section 5.1.

Example 5.8 Consider agents $Q8$ and $Q9$:

$Q8 \stackrel{def}{=} (\nu a, d)\ (a(x).x(w) \mid d(y).\overline{b}c \mid \overline{d}a)$

$Q9 \stackrel{def}{=} (\nu a, d)\ (d(y).\overline{b}c \mid \overline{d}a)$

Applying Rule **[R8]** to agent $Q8$ we obtain agent $Q9$:

Agent		Reduced Agent	Rule
$(\nu a, d)\ (a(x).x(w) \mid d(y).\overline{b}c \mid \overline{d}a)$	$\rightarrow$	$(\nu a, d)\ (d(y).\overline{b}c \mid \overline{d}a)$	**[R8]**

The idea of eliminating codeless π-expression based on inactive actions is captured by rules **[R6]**, **[R7]** and **[R8]**. **[R6]** captures the idea of restricted agents in the prefixing context, while **[R7]** and **[R8]** capture the idea of restricted composed agents in which one of the agents stops because its prefixing action has a restricted channel and cannot communicate with the other agent. These rules, however, do not cover all situations in which composed agents might stop. We might remove, for example, from rule **[R8]**, condition $ch(\beta) \in \vec{x}$ and still have α channel extruded. But this would require a semantic approach to be checked and we avoided such an approach to preserve the rewriting technique (to have this detected, certain communications must be performed due to agents reconfiguration).

6 Conclusions

This paper presented new rewriting rules based on the syntactic characterisation of π-agents active names. The main result is on the application of these

rules to rewriting systems enriching classes of equivalences checked by normalisation: certain composed agents can now be checked bisimilar after agent normalisation. This is a step forward to [3,4] in which only unused prefixed agents could be removed from expressions.

Bisimulation checking can be performed by rewriting rules concerned with structural congruence. There, a normal form for π-expressions is used and agents are bisimilar if they can be rewritten to the same normal form modulo to alpha conversion, commutativity and associativity of parallel composition and permutation of consecutive restrictions. Besides that, Hirschkoff [3,4] developed a technique to extend the idea of checking structural congruence to up-to-bisimulation by Sangiorgi [11,12].

With the syntactic characterisation of active names, we can also find the potential inactive names of agents. From that, we may discover part of expressions never used because it is preceded by names semantically inactive (namely inactive actions). Expressions starting with inactive actions can substitute for stop agent without changing behaviour. In other words, we may use the characterisation of active names to reduce π-expressions and enhance π-*calculus* rewriting verification techniques. In the present work, we investigated how certain unengageable composed agents can be removed without affecting their behaviour. However, this does not cover all situations in which composed agents might stop. To have this detected, certain communications must be performed due to agents reconfiguration and a semantic approach must be applied. New rules on composed agents have been investigated considering the semantic approach, but are still underway as well as rules on replicated agents.

A prototype has been developed to implement the rewriting system by Hirschkoff and the new rules. Rules from **[R1]** to **[R7]** are already implement and **[R8]** is in progress. The system has been successfully applied to toy examples, but no accurate data on performance has been taken so far.

References

[1] W. Rance Cleaveland, editor. *Proceedings of the Second International Conference on Foundations of Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS'99), (Amsterdam, The Netherlands, April 1999)*, volume 1579 of *LNCS*. Springer, 1999.

[2] Ana C. V. de Melo. A study on the potential active names of π-agents. *ENTCS (Electronic Notes in Theoretical Computer Science)*, 95(C):269–286, apr 2004.

[3] D. Hirschkoff. Automatically proving up to bisimulation. In Petr Jancar and Mojmir Kretinsky, editors, *Proceedings of MFCS '98 Workshop on Concurrency*, volume 18 of *ENTCS*. Elsevier Science Publishers, 1998.

[4] D. Hirschkoff. On the benefits of using the up-to techniques for bisimulation verification. In Cleaveland [1], pages 286–299.

[5] B. Jonsson and J. Parrow. Deciding bisimulation equivalences for a class of Non-Finite-State programs. *Information and Computation*, 107(2):272–302, December 1993.

[6] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, May 1999.

[7] U. Montanari and M. Pistore. Checking bisimilarity for finitary π-calculus. In Insup Lee and Scott A. Smolka, editors, *Proceedings of CONCUR '95*, volume 962 of *LNCS*, pages 42–56. Springer, 1995.

[8] Ugo Montanari and Marco Pistore. Finite state verification for the asynchronous pi-calculus. In Cleaveland [1], pages 255–269.

[9] J. Parrow. An introduction to the π-calculus. In Bergstra, Ponse, and Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier, 2001.

[10] M. Pistore and D. Sangiorgi. A partition refinement algorithm for the π-calculus. In Rajeev Alur, editor, *Proceedings of CAV '96*, volume 1102 of *LNCS*, pages 38–49, 1996. Full version to appear in *Journal of Information and Computation*.

[11] D. Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998. An extended abstract appeared in the *Proceedings of MFCS '95*, LNCS 969: 479–488.

[12] D. Sangiorgi and R. Milner. The problem of "weak bisimulation up to". In W. R. Cleaveland, editor, *CONCUR '92: Third International Conference on Concurrency Theory*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46, Stony Brook, New York, 24–27August 1992. Springer-Verlag.