LIF

Laboratoire d'Informatique Fondamentale
de Marseille

Unité Mixte de Recherche 6166
CNRS - Université de Provence - Université de la Méditerranée

On message deliverability and non-uniform receptivity

Roberto M. Amadio, Gérard Boudol, Cédric Lhoussaine

Rapport/Report 05-2002

31 Mai, 2002

On message deliverability and non-uniform receptivity

Roberto M. Amadio (1), Gérard Boudol (2), Cédric Lhoussaine (3)

Laboratoire d'Informatique Fondamentale

UMR 6166
CNRS - Université de Provence - Université de la Méditerranée

(1) Université de Provence, (2) INRIA-Sophia, (3) University of Sussex

amadio@cmi.univ-mrs.fr

Abstract/Résumé

The message deliverability property requires that every emitted message has a chance of being received. In the context of the asynchronous π-calculus, we introduce a discipline of *non-uniform* receptivity that entails this property. Adopting this discipline requires a style of programming where resources are persistent. We give a general method to transform (in a fully abstract way) a process so that it complies with the discipline.

La propriété de livrabilité des messages demande qu'un message émis ait une chance d'être reçu. Dans le contexte du π-calcul asynchrone, nous introduisons une discipline de receptivité *non-uniforme* qui implique cette propriété. Adopter cette discipline demande un style de programmation où les ressources sont persistantes. Nous donnons une methode génerale pour transformer (d'une façon pleinement abstraite) un processus pour qu'il adhère à cette discipline.

Relecteurs/Reviewers: Silvano Dal Zilio, Denis Lugiez.

Notes: The first author is partly supported by IST Profundis. Some of the results presented here were announced in a preliminary form in a paper by the same authors appeared in the proceedings of the FST&TCS'99 Conference, Lecture Notes in Comp. Sci. 1738.

1 Introduction

A process of, say, the *asynchronous* π-calculus (roughly the π-calculus without output prefix) has the message deliverability property if every message emitted in the course of the computation has the possibility of being received later on. This is clearly a desirable property and in computation models that rely so heavily on message exchange it appears to be a fundamental one. One might consider that a programming error occurs whenever a process sends a message to a non-existing or unreachable destination. Note that this problem is particularly acute in the *asynchronous* π-calculus since in this case there is no direct way to check whether a message has been delivered.

Not surprisingly message deliverability is an undecidable property. It is possible to recursively reduce control reachability to message deliverability and the former problem is known to be undecidable even under rather restrictive conditions. For instance, it is undecidable for the asynchronous π-calculus with finite control [5].

From a technical point of view, message deliverability can be regarded as a variant of the standard *liveness* property in Petri Nets requiring that for any transition t and for any reachable configuration m a configuration can be reached from m where t is enabled. Indeed, in the absence of name generation, the asynchronous π-calculus can be reduced to Petri Nets and the decidability proof of the message deliverability property is just a variant of the one for the liveness property.

A natural question is whether we can impose a discipline of programming that guarantees message deliverability while retaining sufficient expressive power. A comparison with traditional type systems may clarify our goals here. There, a desirable property may be that an atom is never applied to an argument. Again this property is undecidable in general but disciplines of programming that ensure this property while retaining sufficient expressive power have been proposed.

Some disciplines may be more interesting than others. For instance, one way to avoid the typing error above is to turn every atom into a function, and one way to ensure message deliverability is to introduce a fake receiver for every channel.

Clearly this discipline is not very satisfying: first it requires no discipline at all, and second it does not preserve the uniqueness of the receiver which is an important property in the distributed framework we aim at [3]. The discipline we advocate here is formalised as a fragment of the π-calculus that we call *receptive* (or π_1^r-calculus for short). In this calculus, one is forced to program with persistent resources which may always react in some way to requests. For instance, the programmer of a service is compelled to code some reaction in any state of this service.

Showing that no expressive power is lost is then a more delicate matter. The kind of receptivity we are looking for must go beyond the 'uniform' and the 'linear' ones [14].[1] Instead the receptivity discipline that we consider is *non-uniform* and can be regarded as a refinement of the π_1-calculus[2] [1] and a typing discipline proposed by Boudol [6] to control the use of resources. We show that the discipline entails message deliverability and we consider its impact on the programming style and the notion of *asynchronous* bisimulation. In particular, we give a fully abstract encoding of the π_1-calculus into the π_1^r-calculus. When combined with previous results on the representation of the join-calculus in the π_1-calculus [2] and on the encoding of the asynchronous π-calculus in the join-calculus [7], our encoding provides a general method to transform a process of the asynchronous π-calculus so that it complies with the receptive discipline.

There is an apparent paradox in these results: on one hand we have a discipline that forces the receptivity of every channel and on the other hand we have a way of reproducing every behaviour (including those that lose messages) while respecting the receptive discipline! The point here is that the receptive discipline forces a style of programming where resources (channels) do not disappear or become inaccessible. However, it cannot rule out certain behaviours where messages are formally received but are processed in a way which is not necessarily interesting for the problem under consideration, *e.g.*, messages are thrown away, resent... For instance, our encoding will not make a process that deadlocks into one that succesfully performs its task (there is no miracle); all it does is to transform the process into a receptive one and the deadlock into a loop where a message is received and resent for ever. We will elaborate further this point in remark 12 once the technical definitions are in place.

---

[1] In the terminology of Sangiorgi, a *uniformly* receptive channel a reacts to a message by activating always the *same* continuation while a *linearly* receptive channel reacts exactly once.

[2] An asynchronous π-calculus with the property that every channel has a unique receiver; hence 1 for *one* receiver.

$$
\begin{array}{lll}
P,\ Q,\ R\ldots \quad ::= & & \text{processes} \\
 & \mathbf{0} & \quad\text{inaction} \\
 \mid & \overline{a}(b_1,\ldots,b_n) & \quad\text{message in channel } a \\
 \mid & a(b_1,\ldots,b_n).P & \quad\text{input on channel } a \\
 \mid & (P \mid Q) & \quad\text{parallel composition} \\
 \mid & [a=b]P,Q & \quad\text{conditional branching} \\
 \mid & (\nu a)\, P & \quad\text{name generation} \\
 \mid & (\mathsf{rec}\, A(a_1,\ldots,a_n).P)(b_1,\ldots,b_n) & \quad\text{recursive parametric process} \\
 \mid & A(a_1,\ldots,a_n) & \quad\text{recursive parametric call}
\end{array}
$$

$$
(out)\ \frac{}{\overline{a}\vec{b} \xrightarrow{\overline{a}\vec{b}} \mathbf{0}}
\qquad\qquad
(in)\ \frac{}{a(\vec{b}).P \xrightarrow{a\vec{c}} [\vec{c}/\vec{b}]P}
$$

$$
(ext)\ \frac{P \xrightarrow{(\nu\vec{c})\,\overline{b}\vec{b}} P' \quad a \neq b, a \in \{\vec{b}\}\backslash\{\vec{c}\}}{(\nu a)\, P \xrightarrow{(\nu a,\vec{c})\,\overline{b}\vec{b}} P'}
\qquad\qquad
(\nu)\ \frac{P \xrightarrow{\alpha} P' \quad a \notin n(\alpha)}{(\nu a)\, P \xrightarrow{\alpha} (\nu a)\, P'}
$$

$$
(cm)\ \frac{P \xrightarrow{(\nu\vec{c})\,\overline{a}\vec{b}} P' \quad Q \xrightarrow{a\vec{b}} Q' \quad \{\vec{c}\} \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\tau} (\nu\vec{c})\, (P' \mid Q')}
\qquad\qquad
(cp)\ \frac{P \xrightarrow{\alpha} P' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}
$$

$$
(m_t)\ \frac{P \xrightarrow{\alpha} P'}{[a=a]P,Q \xrightarrow{\alpha} P'}
\qquad\qquad
(m_f)\ \frac{Q \xrightarrow{\alpha} Q' \quad a \neq b}{[a=b]P,Q \xrightarrow{\alpha} Q'}
$$

$$
(rec)\ \frac{[\mathsf{rec}\, A(\vec{b}).P/A, \vec{c}/\vec{b}]P \xrightarrow{\alpha} P'}{(\mathsf{rec}\, A(\vec{b}).P)(\vec{c}) \xrightarrow{\alpha} P'}
\qquad\qquad
(id)\ \frac{P = P' \quad P' \xrightarrow{\alpha} Q' \quad Q' = Q}{P \xrightarrow{\alpha} Q}
$$

Figure 1: Asynchronous π-calculus and its labelled transition system

2 Message deliverability

We fix our notation for the asynchronous polyadic π-calculus. Sorts are defined by the following grammar:

$$s ::= val \mid Ch(s_1,\ldots,s_n) \tag{1}$$

We assume a set $\mathcal{N}$ of *names*, ranged over by a, b, $c,\ldots$ and suppose that (i) every name a comes with a fixed sort $st(a) = s$ and that (ii) for every sort there are denumerable many names of that sort. We also assume a denumerable set of *parametric process identifiers* $A, B, \ldots$ and suppose that every identifier A comes with a fixed sort $st(A) = Ch(s_1,\ldots,s_n)$ and an *input arity* $ia(A) = k$ so that n is the number of parameters and $k \leq n$. The input arity will only play a role later in the context of the π_1-calculus. In substitutions, a name (an identifier) can only be replaced by a name (an identifier) with the same sort (same sort and input arity).

Processes, with the usual labelled transitions system, are given in figure 1 where $=$ stands for α-renaming. The symmetric rules for *(cm)* and *(cp)* are omitted. Conventionally, we set $n(\alpha) = fn(\alpha) \cup bn(\alpha)$ where $fn(\tau) = \emptyset$, $fn(a\vec{b}) = \{a\} \cup \{\vec{b}\}$, $fn((\nu\vec{c})\,\overline{a}\vec{b}) = \{a, \vec{b}\}\backslash\{\vec{c}\}$, and $bn(\tau) = bn(\overline{a}\vec{b}) = \emptyset$, $bn((\nu\{\vec{c}\})\,\overline{a}\vec{b}) = \{\vec{c}\}$. Moreover, $fn(P)$ ($bn(P)$) stands for the names occurring free (bound) in P.

In a process we assume that: (i) the formal parameters in $a(\vec{b}).P$ or $(\mathsf{rec}\, A(\vec{a}).P)$ are all distinct. (ii) in $(\mathsf{rec}\, A(\vec{a}).P)$, $fn(P) \subseteq \{\vec{a}\}$, (iii) all process identifiers are bound by a recursive definition, and (iv) recursion is *guarded*, that is in $(\mathsf{rec}\, A(\vec{a}).P)(\vec{b})$ all recursive calls to A in P occur under an input guard.

We denote with SP the result of the application of a substitution S acting on names to a process P.

Henceforth we will only consider *well-sorted* processes. This is the least class of processes such that $\mathbf{0}$ is well sorted and if P, Q are well sorted then:

- $\overline{a}(b_1,\ldots,b_n)$ and $a(b_1,\ldots,b_n).P$ are well sorted if $st(a) = Ch(s_1,\ldots,s_n)$ and $st(b_i) = s_i$ for $i = 1,\ldots,n$.
- $P \mid Q$ and $(\nu a)\, P$ are well sorted.

- $[a = b]P, Q$ is well sorted if $st(a) = st(b) = val$ (so we only allow testing values for equality).
- $(\mathsf{rec}\, A(a_1, \ldots, a_n).P)(b_1, \ldots, b_n)$ is well sorted if $st(A) = Ch(s_1, \ldots, s_n)$ and $st(a_i) = st(b_i) = s_i$ for $i = 1, \ldots, n$.
- $A(b_1, \ldots, b_n)$ is well sorted if $st(A) = Ch(s_1, \ldots, s_n)$ and $st(b_i) = s_i$ for $i = 1, \ldots, n$.

2.1 Asynchronous bisimulation

We recall the notion of asynchronous bisimulation and some related proof techniques.

Definition 1 (bisimulation) *A relation $\mathcal{R}$ on well-formed processes is a bisimulation if it is symmetric, and if $P\,\mathcal{R}\,Q$ implies:*
(1) *if $P \xrightarrow{\tau} P'$ then $Q \xrightarrow{\tau} Q'$ for some Q' such that $P'\,\mathcal{R}\,Q'$,*
(2) *if $P \xrightarrow{(\nu \vec{c})\overline{a}\vec{b}} P'$ and $\{\vec{c}\} \cap fn(Q) = \emptyset$, then $Q \xrightarrow{(\nu \vec{c})\overline{a}\vec{b}} Q'$ for some Q' such that $P'\,\mathcal{R}\,Q'$,*
(3) *if $P \xrightarrow{a\vec{b}} P'$ then there exists Q' such that either $Q \xrightarrow{a\vec{b}} Q'$ and $P'\,\mathcal{R}\,Q'$, or $Q \xrightarrow{\tau} Q'$ and $P'\,\mathcal{R}\,(Q' \mid \overline{a}(\vec{b}))$.*

We denote with $\sim$ the largest bisimulation. The notion of *weak* bisimulation is obtained by replacing everywhere transitions $\xrightarrow{\alpha}$ with weak transitions $\xRightarrow{\alpha}$ defined as usual, that is $\xRightarrow{\alpha} = (\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^*$ if $\alpha \neq \tau$, and $\xRightarrow{\tau} = (\xrightarrow{\tau})^*$. We denote with $\approx$ the largest weak bisimulation, and by $\mathcal{F}$ the related monotonic operator on binary relations which has $\approx$ as largest fixed point. We recall the following basic properties of (asynchronous) bisimulation. The proof given in [4] requires some substantial case analysis for the input transition.

Proposition 2 (1) *The relation $\approx$ is an equivalence relation.*
(2) *If $P \approx Q$ then $(P \mid \overline{a}(\vec{b})) \approx (Q \mid \overline{a}(\vec{b}))$.*

We have the standard fact that we get the same relation $\approx$ if in the definition of weak bisimulation we require that a strong transition is matched by a weak one. We also need a few notions of *bisimulation up to* techniques. To this end, we define the partial order $\leq_{\mathcal{F}}$ on relations as

$$\mathcal{R}_1 \leq_{\mathcal{F}} \mathcal{R}_2 \quad \text{iff} \quad \mathcal{R}_1 \subseteq \mathcal{R}_2 \quad \text{and}\ \mathcal{R}_1 \subseteq \mathcal{F}(\mathcal{R}_2)\ .$$

Let H be an operator on binary relations that is monotonic with respect to $\leq_{\mathcal{F}}$ – we also say that H preserves $\leq_{\mathcal{F}}$ in this case[3]. Then we say that $\mathcal{R}$ is a *weak bisimulation up to H* if $\mathcal{R} \subseteq \mathcal{F}(H(\mathcal{R}))$. The following result is proved in [15].

Proposition 3 (1) *The family of operators preserving $\leq_{\mathcal{F}}$ is closed under composition and union.*
(2) *If an operator H on binary relations preserves $\leq_{\mathcal{F}}$ and $\mathcal{R}$ is a weak bisimulation up to H then $\mathcal{R} \subseteq \approx$.*

The proof of (1) is immediate and the proof of (2) amounts to build a weak bisimulation, starting from $\mathcal{R}$ and iterating the operator $\lambda\mathcal{S}.(\mathcal{S} \cup H(\mathcal{S}))$.

2.2 Message deliverability property

First we need to fix some technical notions. An *evaluation context* E is defined by

$$E ::= [\,] \mid E \mid P \mid P \mid E \mid (\nu a)\, E \tag{2}$$

We introduce the following *abbreviations*:

$$T_a = (\mathsf{rec}\, A(a).a(\vec{b}).A(a))(a), \quad Id_a = (\mathsf{rec}\, A(a).a(\vec{b}).(\overline{a}\vec{b} \mid A(a)))(a),$$

$$a(\vec{b}) : P = a(\vec{b}).(P \mid T_a)\ .$$

The *terminated* process T_a repeatedly receives and throws away messages on channel a, the *identity* process Id_a repeatedly receives and sends back messages on channel a, the *input once* prefix operator ':' receives *once* a message on a channel, say a, and then spawns a process T_a in parallel with the continuation.

[3] In [15], an operator that preserves $\leq_{\mathcal{F}}$ is called 'respectful'.

It is convenient to consider processes up to a *structural equivalence* $\cong$ defined as the least equivalence relation such that:

$$
\begin{array}{llll}
(1) & P \mid Q \cong Q \mid P & (2) & (P \mid Q) \mid R \cong P \mid (Q \mid R) \\
(3) & P \mid \mathbf{0} \cong P & (4) & (\nu a)\, P \mid Q \cong (\nu a)\,(P \mid Q) \text{ if } a \notin fn(Q) \\
(5) & (\nu a)\,(\nu b)\, P \cong (\nu b)\,(\nu a)\, P & (6) & (\mathsf{rec}\, A(\vec{b}).P)(\vec{c}) \cong [\mathsf{rec}\, A(\vec{b}).P/A, \vec{c}/\vec{b}]P \\
(7) & [a = a]P, Q \cong P & (8) & [a = b]P, Q \cong Q \text{ if } a \neq b \\
(9) & (\nu a)\, P \cong P \text{ if } a \notin fn(P), st(a) = val & (10) & (\nu a)\, T_a \cong \mathbf{0} \\
(11) & E[P] \cong E[Q] \text{ if } P \cong Q\ . & &
\end{array}
$$

Equations (1–5) are standard. Equations (6–8) are about unfolding and branching; they are employed to get rid of internal deterministic transitions and to obtain a nice canonical form (lemma 4(2)). Of course equation (8) is not compatible with arbitrary contexts and we just require closure under evaluation contexts (rule (11)). Equations (9–10) are non-standard; they garbage collect some dead names and processes and are employed in the proofs of section 4.2. As stated below structurally equivalent processes are strongly bisimilar.

Lemma 4 (1) *If $P \cong Q$ then $P \sim Q$.*
(2) *Any process P is structurally equivalent to a process of the shape*

$$(\nu \vec{c})\,(\Pi_{i\in I}\overline{a}_i(\vec{a}_i) \mid \Pi_{j\in J} b_j(\vec{b}_j).P_j) \tag{3}$$

where $\{\vec{c}\}$, I, and J can be empty, Π stands for the parallel composition of a family of processes, and we conventionally take the parallel composition of an empty family to be $\mathbf{0}$.

Thus we arrive at the definition of the message deliverability property.

Definition 5 (1) *We write $P \downarrow a$ if P offers a visible input on a,* i.e., *$P \stackrel{a\vec{b}}{\rightarrow} P'$ for some $\vec{b}, P'$.*
(2) *We say that a process P has the* message deliverability property, *if whenever $P \stackrel{\tau}{\Rightarrow} Q$ with $Q \cong (\nu \vec{a})\,(\overline{a}\vec{b} \mid P')$ then $P' \stackrel{\tau}{\Rightarrow} P''$ and $P'' \downarrow a$.*

In principle, we can always transform a process into a bisimilar one having the message deliverability property. The method is to introduce for every channel a an identity process Id_a.

Proposition 6 *Given a process P, we can effectively build a process P' which has the message deliverability property and is equivalent to P up to weak asynchronous bisimulation.*

Proof hint. Let $fch(P) = \{a \mid a \in fn(P) \text{ and } st(a) = Ch(\vec{s})\}$. Define $P' = I(P) \mid \Pi_{a\in fch(P)} Id_a$, where I is a function that commutes with every process constructor but the channel generator where it is defined by:

$$I((\nu a)\, P) = (\nu a)\,(I(P) \mid Id_a) \quad \text{if } st(a) = Ch(\vec{s})\ .$$

Then show that (1) P' has the message deliverability property and (2) $P \approx P'$. □

As mentioned in the introduction, this transformation is not very satisfying; in particular it introduces the identity process also when it is not needed. We anticipate that this transformation is ruled out in π_1 (and π_1^r) by the requirement that the receiver is *unique*.

Next we pause to consider: (i) the decidability of the message deliverability property and (ii) its connection with the classical *liveness* property for Petri Nets mentioned in the introduction.

Concerning (i), we recall that the *control reachability problem* amounts to dermine, given a process P, whether P can reach a specific point of the control determined by, say, a special constant A, *i.e.*, whether $P \stackrel{\tau}{\Rightarrow} P'$ and $P' \cong (\nu \vec{a})\,(A \mid P'')$ for some $\vec{a}, P''$. It has been shown in [5] that the halting problem for 2-counter machines can be recursively reduced to the control reachability problem for two fragments of the asynchronous π-calculus that combine 'name generation' with either 'name mobility' or 'unbounded' control. Roughly, 'name generation' is the possibility of generating fresh names (values or channels), name mobility is the possibility of transmitting names, and unbounded control is the possibility of dynamically adding new threads of control. The following proposition 7(1) gives a recursive reduction of the control reachability problem to the message deliverability problem and therefore it proves the undecidability of the latter.

Concerning (ii), we rely on a well-known encoding of the asynchronous π-calculus *without* name generation into Petri Nets that basically goes back to early work [8] on the translation of ccs to Petri Nets. In the encoding,

$$(\nu_{val}) \quad \frac{I \vdash P \quad st(a) = val \quad a \notin I}{I \vdash (\nu a)\, P} \qquad (\nu_{Ch}) \quad \frac{I \cup \{a\} \vdash P \quad st(a) = Ch(\vec{s}) \quad a \notin I}{I \vdash (\nu a)\, P}$$

$$(out) \quad \frac{}{\emptyset \vdash \overline{a}\vec{b}} \qquad (in) \quad \frac{\{a\} \cup I \vdash P \quad \{\vec{b}\} \cap I = \emptyset}{\{a\} \cup I \vdash a(\vec{b}).P}$$

$$(|) \quad \frac{I_j \vdash P_j \quad j = 1,2 \quad I_1 \cap I_2 = \emptyset}{I_1 \cup I_2 \vdash P_1 \mid P_2} \qquad (=) \quad \frac{I \vdash P_i, \ i = 1,2}{I \vdash [a = b]P_1, P_2}$$

$$(\mathbf{0}) \quad \frac{}{\emptyset \vdash \mathbf{0}} \qquad (\mathsf{rec}_1) \quad \frac{\sharp\{b_1, \ldots, b_k\} = ia(A)}{\{b_1, \ldots, b_k\} \vdash A(b_1, \ldots, b_n)}$$

$$(\mathsf{rec}_2) \quad \frac{\{a_1, \ldots, a_k\} \vdash P \quad \sharp\{b_1, \ldots, b_k\} = ia(A)}{\{b_1, \ldots, b_k\} \vdash (\mathsf{rec}\, A(a_1, \ldots, a_n).P)(b_1, \ldots, b_n)}$$

Figure 2: Interface

messages and control points are represented by tokens in certain (distinct) places. The message deliverability property then requires that the tokens representing messages have a chance of being consumed, *i.e.*, that a certain transition t can be fired. This property recalls the *liveness* property and indeed the decidability proof for the latter (see, *e.g.*, [13]) can be easily adapted to the former (proposition 7(2)).

Proposition 7 (1) *The control reachability problem is recursively reducible to the message deliverability problem.*

(2) *In the absence of name generation, the message deliverability problem is recursively reducible to the reachability problem for Petri Nets.*

Proof. (1) Suppose we want to decide whether the process P reaches a control point A. We transform P into P' as in the proof of proposition 6. Then P' has the message deliverability property and clearly P reaches A iff P' reaches A. Now turn the control point A into a fresh channel and consider the process $P'' = \overline{A} \mid !((\nu\vec{a})\,[A.\mathbf{0}/A]P')$ where ! is the usual replication operator and $\{\vec{a}\}$ are the names free in $[A.\mathbf{0}/A]P'$ but A. Then P'' satisfies the message deliverability property iff P' reaches A iff P reaches A.

(2) See appendix A. $\square$

3 Non-uniform receptivity

An *interface* I is a *set* of names of channel sort. We introduce in figure 2 a formal system to determine when a well-sorted process has interface I (written $I \vdash P$). Intuitively, if $I \vdash P$ then P may perform inputs on the channels in the interface I. Here the input arity of a process identifier plays a role: it declares the parameters on which an input can be performed. In particular, the first $ia(A)$ parameters of an identifier A must be of channel sort.

It is easy to check that a process P has at most one interface I. Moreover if $I \vdash P$ then there is at most one thread that can perform an input on a given channel (unique receiver property). We note that to achieve this property the system requires: (i) not to receive on received channels (cf. work on the *local* π-calculus and channels with *output capability*, see, *e.g.*, [11]), (ii) disjoint interfaces of parallel processes (condition $I_1 \cap I_2 = \emptyset$ in rule (|)), and (iii) injective instanciation of the first $ia(A)$ parameters of an identifier A.

We write $I \vdash^r P$ (r for receptive) if $I \vdash P$ and moreover: (1) if A occurs in P then $ia(A) = 1$, and (2) in all applications of the input rule (in), the interface I is empty. Intuitively, if $I \vdash^r P$ and $a \in I$ then P is always ready to perform an input on a. For instance, $\{a\} \vdash^r T_a$, $\{a\} \vdash^r Id_a$, and $\{a\} \vdash^r a(\vec{b}) : P$ if $\emptyset \vdash^r P$. On the other hand, if we set $P = (\mathsf{rec}\, A(a,b).a.b.A(a,b))(a,b)$ then $\{a,b\} \vdash P$ but $\{a,b\} \not\vdash^r P$.

Both notions of interface are preserved by labelled transitions.

Proposition 8 (subject reduction) *Suppose $I \vdash P$ and $P \xrightarrow{\alpha} P'$. Then:*

(1) *$\alpha \equiv \tau$ or $\alpha \equiv a\vec{b}$ implies $I \vdash P'$.*

(2) *$\alpha \equiv (\nu\vec{c})\,\overline{a}\vec{b}$ and $\vec{c'}$ names in $\vec{c}$ of channel sort implies $I \cup \{\vec{c'}\} \vdash P'$.*

The same holds if we replace everywhere $\vdash$ by $\vdash^r$.

Proof hint. The proof is a variant of the one presented in [2]. First show that if $I \vdash P$ then, for any substitution S injective on I, $S(I \vdash P)$. Then proceed by induction on the derivation of $P \xrightarrow{\alpha} P'$. The only difficulty arises with the unfolding of the recursion $(\mathsf{rec}\, A(\vec{a}).P)(\vec{c})$. In this case one shows that if $ia(A) = k$, $\{a_1, \ldots, a_k\} \vdash P$, and $\sharp\{c_1, \ldots, c_k\} = k$ then $\{c_1, \ldots, c_k\} \vdash [\mathsf{rec}\, A(\vec{a}).P/A, \vec{c}/\vec{a}]P$. □

We define π_1 and π_1^r as follows: the π_1-calculus is composed of the processes P such that $I \vdash P$ and the π_1^r-calculus of the processes P such that $I \vdash^r P$. It is intended that in both π_1 and π_1^r the notion of structural equivalence introduced in section 2.2 relates only processes with the *same* interface. The receptive system $\vdash^r$ has the following additional (and announced) property.

Proposition 9 (receptivity) *Suppose $I \vdash^r P$. Then:*

(1) *$P \downarrow a$ iff $a \in I$.*

(2) *If $P \downarrow a$ and $P \xrightarrow{\tau} P'$ then $P' \downarrow a$.*

Proof. (1) ($\Rightarrow$) We proceed by induction on the inference of $P \downarrow a$. ($\Leftarrow$) By induction on the inference of $I \vdash^r P$.

(2) By (1) and the subject reduction proposition 8(1). □

We note that if $I \vdash^r (\nu c)\, P$ and c is of channel sort then $I \cup \{c\} \vdash^r P$ and therefore P includes a persistent receiver for c. From this, we derive message deliverability under suitable conditions.

Corollary 10 (message deliverability) *If $I \vdash^r P$ and all free channels in P are in I then P has the message deliverability property.*

Proof. Suppose $I \vdash^r P$ and $P \stackrel{\tau}{\Rightarrow} P' \cong (\nu\vec{a})\,(\overline{a}\vec{b} \mid P'')$. Then $I \vdash^r P'$ by proposition 9(2). Let $\{\vec{a'}\}$ be the set of channels in $\{\vec{a}\}$. By definition of $\vdash^r$ we know that $a \in I \cup \{\vec{a'}\}$ and that $I \cup \{\vec{a'}\} \vdash^r \overline{a}\vec{b} \mid P''$. By definition of $\vdash^r$ it follows that $I \cup \{\vec{a'}\} \vdash^r P''$ and by proposition 9(1, $\Leftarrow$) we conclude that $P'' \downarrow a$. □

Remark 11 (1) *Note that in π_1^r we have a* strong *form of message deliverability: if we reach a process $(\nu\vec{a})\,(\overline{a}\vec{b} \mid P'')$ then P'' can offer* immediately *(essentially up to unfolding and branching) an input on channel a.*

(2) *Both calculi can be enriched with a notion of* linear *channel in the sense of [9]. The distinctive property of a linear channel is that it is used exactly once in the course of the computation. A typical example being the 'return' channel r of a 'remote procedure call' $(\nu r)\,(\overline{a}(r, v) \mid r(u) : P)$ (an instance of this schema is found in the encoding described in section 4.1). The extension is not too difficult to write down but it introduces some notation and case analysis that seems better to avoid to convey the essence of our contribution.*

Next we turn to the notion of *bisimulation*. The asynchronous variant recalled in definition 1 is amended as follows for both π_1 and π_1^r:

(1) we observe an output transition $P \xrightarrow{(\nu\vec{c})\overline{a}\vec{b}} P'$ only if the channel a is not in the interface of P.

(2) if $\mathcal{R}$ is a bisimulation and $P \,\mathcal{R}\, Q$ then P and Q have the *same* interface.

The first condition comes naturally from the fact that we require the *unicity of the receiver*: if the receiver is defined in the *observed* process then it cannot be defined in the *observer*! The second condition is a corollary of the first one: if $I \vdash P$ and $I' \vdash Q$ and $a \in I \backslash I'$ then in $P \mid \overline{a}\vec{b}$ we cannot observe the output $\overline{a}\vec{b}$ while in $Q \mid \overline{a}\vec{b}$ we can. Thus, if our bisimulation has to be preserved by parallel composition then it cannot relate processes with different interfaces.

A consequence of (1) is that $Id_a \approx T_a$ (a bisimulation is easily built). A consequence of (2) is that $Id_a \not\approx \mathbf{0}$ since these two processes do not have the same interface. Because of condition (1), on processes with the same interface the amended notion of bisimulation provides a *coarser* notion of equivalence. This is instrumental to show, *e.g.*, that the joined input of the join-calculus can be defined in the π_1-calculus up to weak asynchronous bisimulation [2].

On the other hand, it has been shown in [7] that there is a fully abstract encoding of the asynchronous π-calculus in the join-calculus. These two results provide evidence for the expressivity of the π_1-calculus.

What about the expressivity of the π_1^r-calculus? We consider two examples that illustrate the style of 'programming' that one must adopt to conform to the receptive discipline. We denote by $\overline{a}(\vec{v}, \_, \vec{v'})$ the term $(\nu b)\,(\overline{a}(\vec{v}, b, \vec{v'}) \mid T_b)$ if b is a channel and $(\nu b)\,\overline{a}(\vec{v}, b, \vec{v'})$ if b is a value. We write a recursive process $(\mathsf{rec}\, A(\vec{a}).P)(\vec{a})$ that does not introduce new parameters simply as $\mathsf{rec}\, A(\vec{a}).P$. Finally, we define a *replicated* input given by $a^*(\vec{u}).P \;=\; \mathsf{rec}\, A(a).a(\vec{u}).(P \mid A(a))$.

Buffers A typical non-receptive agent is the 'one-slot buffer' that repeatedly waits for some data on a given channel and then sends it on another channel. In the synchronous π-calculus, this process may be written:

$$\big(\mathsf{rec}\, B(a,b).a(c).\overline{b}(c).B(a,b)\big)(a,b)$$

Clearly, this cannot be written so easily in π_1^r. In π_1^r we program the one-slot buffer as follows: first it inputs on a a message that is supposed to convey a datum to store in the buffer (if this is not the case the message is ignored, i.e, it is resent), and then on the *same* channel it receives a request for extracting the contents of the buffer, which is delivered on a private return channel (again, if this protocol is violated this second message is ignored, though obviously something more elaborate could be done in a more synchronous version of the buffer).

$$\begin{aligned} \mathit{Buff}_1(a) = \mathsf{rec}\, B(a).a(k_1,x,y).[k_1 \neq \mathsf{put}]&(\overline{a}(k_1,x,y) \mid B(a)),\\ \mathsf{rec}\, B_1(a,x).a(k_2,z,y).[k_2 \neq \mathsf{get}]&(\overline{a}(k_2,z,y) \mid B_1(a,x)),\\ &\overline{y}(x) \mid B(a) \end{aligned}$$

It is easily checked that (i) $\mathit{Buff}_1(a)$ is well-sorted assuming, *e.g.*, that the content of the buffer is of value type, and (ii) that $\{a\} \vdash^r \mathit{Buff}_1(a)$. The requests for reading and writing the buffer are respectively $(\nu c)\,(\overline{a}(\mathsf{get}, \_, c) \mid c(x){:}P)$ and $\overline{a}(\mathsf{put}, b, \_)$. As one can see, the buffer is now a kind of 'agent', that is a process which is invoked by its name a and reacts according to some internal protocol. To build a 'two-slots buffer' from this one, we may proceed as usual, putting together two one-slot buffers with a private communication between them. However, to write this we need to refine our previous program, because we need to explicitly indicate the keys used, defining $\mathit{Buff}_1(a, \mathsf{put}, \mathsf{get})$ – in the obvious way. This is left as an exercise for the reader.

Mutual exclusion Synchronization can be 'programmed' in the π-calculus, a typical example being mutual exclusion between tasks enforced by the use of a *lock*, as follows:

$$(\nu l)\,(\,\overline{l} \;\mid\; \mathsf{task}_1^*(\vec{a}_1).l().(\,\cdots\,\overline{l}\,) \;\mid \ldots\; \mid \mathsf{task}_n^*(\vec{a}_n).l().(\,\cdots\,\overline{l}\,)\,)\;.$$

This violates both receptivity and the unique receiver property. In the receptive style, the lock is represented by a process that receives on a unique channel l messages carrying a value lock or unlock and a return channel r:

$$\begin{aligned} \mathsf{Lock}(l) \;=\; \mathsf{rec}\, A(l).l(k,r).[k \neq \mathsf{lock}]&(\overline{l}(k,r) \mid A(a)),\\ \overline{r} \mid \mathsf{rec}\, A'(l).l(k',s).[k' \neq \mathsf{unlock}]&(\overline{l}(k',s) \mid A'(l)),\\ &A(l) \end{aligned}$$

and a task is now written $\mathsf{task}_i^*(\vec{a}_i).(\nu r)\,\big(\overline{l}(\mathsf{lock}, r) \;\mid\; r(){:}(\,\cdots\,\overline{l}(\mathsf{unlock}, \_))\big)$. Again, the lock is a persistent agent that has an identity l, and reacts according to its own protocol, governed by the keys it receives.

Remark 12 (on busy waiting) *As the reader might have already noticed, in both examples we enforce receptivity by introducing a form of* busy waiting, i.e., *received messages with the 'wrong' pattern are immediately resent. This trade-off between receptivity and busy waiting seems hard to avoid and it will be found again in the general encoding of π_1 into π_1^r (see the implementation of the* channel manager *in section 4.3). This should not come as a surprise; we cannot expect our encoding to be so clever as to make a process that loses messages into a 'correct' one that does not. Indeed, in a more* intentional *sense, the encoded process can still lose messages by busy waiting. All the encoding shows is that, in a sense, every behaviour of π_1 can be programmed in π_1^r. It is not quite our intention to advocate programming with busy waiting as a good programming style, but we note* *that in practice there might be more clever ways of handling unwanted messages. For instance, in the example of the buffer one could send back a message of the kind 'sorry buffer full (or empty); try later'. Of course, this kind of transformations require some understanding of the problem at hand and they can only be* encouraged *by the receptive discipline.*

4 From π_1 to π_1^r

We show that there is a fully abstract encoding of π_1 into π_1^r. One basic problem is that in general the channels on which a thread of the π_1-calculs will perform an input cannot be statically determined. Thus the encoding methods employed in the previous two examples do not apply directly.

4.1 Encoding

We use the notation $[C]P, Q$ when C is a boolean combination of name equalities $a = b$ and inequalities $a \neq b$. This notation is compiled in the obvious way:

$$[\neg C]P, Q = [C]Q, P \quad [C \vee C']P, Q = [C]P, ([C']P, Q) \quad [C \wedge C']P, Q = [C]([C']P, Q), Q$$

We use a default channel notation '_' in other contexts besides output:

(i) in an input, $a(\vec{b}, \_, \vec{b'}).P$ denotes $a(\vec{b}, c, \vec{b'}).P$ where $c \notin fn(P)$,
(ii) in a recursive call $A(\_, \vec{b})$ stands for $(\nu a)\, A(a, \vec{b})$.
(iii) in a recursive definition, $(\mathsf{rec}\, A(\_, \vec{b}).P)$ denotes $(\mathsf{rec}\, A(a, \vec{b}).(T_a \mid P))$.

In the proofs we also use the notation $P \stackrel{(\nu\vec{c})\,\overline{a}\vec{b},\_,\vec{b'}}{\longrightarrow} P'$, which stands for $P \stackrel{(\nu\vec{c},c)\,\overline{a}\vec{b},c,\vec{b'}}{\longrightarrow} P'$ where $c \notin fn(P)$ and $st(c) = val$.

Now the idea of the encoding is rather simple: we turn any message on a channel a into a request to a *channel manager* $CM(a)$ for a, sending the arguments of the message together with a key *out*. Symmetrically, we turn any input on a into a request to $CM(a)$, sending a key *in* and a private return channel to actually receive something. The channel manager will filter the messages according to the keys, and act as appropriate.

Let us pause to note that the use of a channel manager is not new. For instance, it is similar to what is done in implementing communication for a language like the π-calculus except that one would exploit elaborate data structures like 'pools' or queues to manage the input and output requests in a more realistic way. On a technical level, a notion of channel manager occurs in the encoding of the asynchronous π-calculus in the join-calculus [7]. That channel manager has a simpler definition since it does not have to be receptive.

Going back to our encoding, we note that there is an attack which compromises abstraction: the environment can send a request for input to the channel manager. This requires an additional twist: we authenticate the requests for input by introducing a restricted key in_a for every channel manager which is known only by the process that can actually input on that channel.

Formally, for every name a we assume a fresh name in_a (that is not in the names of the translated processes). The name in_a has sort *val* and it is used as the key of the channel a. We translate a term P of the π_1-calculus with interface I, where $I = \{a_1, \ldots, a_n\}$, into the following process

$$\langle\!| I, P |\!\rangle = (\nu in_{a_1}) \cdots (\nu in_{a_n})\, (CM(a_1, in_{a_1}) \mid \cdots \mid CM(a_n, in_{a_n}) \mid [\![P]\!])$$

where $[\![P]\!]$ is defined below, which turns out to be also a well-formed process of the π_1^r-calculus with interface $\emptyset$. Sorts are translated as follows:

$$\begin{array}{rcl} [\![val]\!] & = & val \\ [\![Ch(s_1, \ldots, s_n)]\!] & = & Ch(val, [\![s_1]\!], \ldots, [\![s_n]\!], Ch([\![s_1]\!] \ldots, [\![s_n]\!]), val, val) \end{array}$$

where the first argument is the input/output key of the channel, then come the arguments of the message to be delivered, followed by the type of the return channel to which they are actually sent, and then we have two keys for internal choice. The following transition rules describe the behaviour of the channel manager where we assume $j_1 \neq in_a$:

$$(CM_\tau) \quad (CM(a, in_a) \mid \overline{a}(j_1, \vec{b}_1, r_1, c_1, c_1') \mid \overline{a}(in_a, \vec{b}_2, r_2, c_2, c_2')) \stackrel{\tau}{\rightarrow} (CM(a, in_a) \mid \overline{r_2}(\vec{b}_1))$$

$$(CM_{in}) \quad (CM(a, in_a) \mid \overline{a}(in_a, \vec{b}_2, r_2, c_2, c_2')) \stackrel{a(j_1, \vec{b}_1, r_1, c_1, c_1')}{\longrightarrow} (CM(a, in_a) \mid \overline{r_2}(\vec{b}_1))$$

By this specification, the channel manager matches a request for input with a request for output (the latter can be provided by the environment). A reduction such as (CM_τ) could be directly implemented in a join-calculus enriched with a filter condition on the received messages. However, to simplify the proofs we will in a first step reason with the axiomatic specification above. That is, we extend the π_1^r with a new constant $CM(a, in_a)$ (with two free parameters), which behaves as prescribed and is such that $\{a\} \vdash^r CM(a, in_a)$. Then we will see how to implement the channel manager in the π_1^r-calculus, up to weak bisimulation. In order to be able to use the specification of CM, we must add a transition rule that allows structural manipulations to be performed:

$$P \xrightarrow{\alpha} P' \text{ and } Q \cong P \;\Rightarrow\; Q \xrightarrow{\alpha} P'$$

In the π_1^r this is harmless since $\cong$ is a strong bisimulation. The encoding $[\![P]\!]$ of processes is as follows:

$$\begin{array}{rcl}
[\![\mathbf{0}]\!] & = & \mathbf{0} \\
[\![\overline{a}\vec{b}]\!] & = & \overline{a}(_, \vec{b}, _, _, _) \\
[\![a(\vec{b}).P]\!] & = & (\nu r)\,(\overline{a}(in_a, _, r, _, _) \mid r(\vec{b}){:}[\![P]\!]) \\
[\![P \mid Q]\!] & = & ([\![P]\!] \mid [\![Q]\!]) \\
[\![[a = b]P, Q]\!] & = & [a = b][\![P]\!], [\![Q]\!] \\
[\![(\nu a)\, P]\!] & = & (\nu a)\,(\nu in_a)\,(CM(a, in_a) \mid [\![P]\!]) \text{ if } st(a) = Ch(\vec{s}) \\
[\![(\nu a)\, P]\!] & = & (\nu a)\,[\![P]\!] \text{ if } st(a) = val
\end{array}$$

Regarding the encoding of recursion, we assume given an injection that maps identifiers A with arity $ia(A) = k$ of π_1 into identifiers of π_1^r with arity 1. For simplicity, we keep the same name and map A to A. Moreover, assume $st(A) = Ch(s_1, \ldots, s_n)$ and suppose that:

$$\vec{b} \equiv b_1, \ldots, b_n \equiv a_1, \ldots, a_k, c_{k+1}, \ldots, c_n \equiv \vec{a}, \vec{c}\,.$$

and similarly for $\vec{b}'$. Then we define:

$$\begin{array}{rcl}
[\![A(\vec{b})]\!] & = & A(_, \vec{a}, \vec{in}_a, \vec{c}) \\
[\![(\mathsf{rec}\, A(\vec{b}).P)(\vec{b}')]\!] & = & (\mathsf{rec}\, A(_, \vec{a}, \vec{in}_a, \vec{c}).[\![P]\!])(_, \vec{a}', \vec{in}_{a'}, \vec{c}')
\end{array}$$

Remark 13 *We note that the only receivers in the encoding are the channel managers and the 'input once' return channels r. It is clear from the specification of the channel manager and later from its implementation, that* at most one *message is emitted on r. Thus r can be regarded as a weakly linear (or* affine*) channel.*[4] *Moreover, an encoded process is composed of a set of servers (the channel managers) and a bunch of processes that keep performing remote procedure calls on the servers and possibly create new ones.*

As expected, the encoding preserves the interfaces.

Proposition 14 *If $I \vdash P$ in the π_1-calculus then $\emptyset \vdash^r [\![P]\!]$ and $I \vdash^r \langle\!| I, P |\!\rangle$.*

PROOF. By induction on the definition of $I \vdash P$. □

4.2 Definition of the bisimulation relation

Now we embark on the proof that our encoding is fully abstract with respect to weak bisimulation. Our proof relies on the bisimulation up to technique (cf. proposition 3). As pointed out in the following remark 17, the application of the theory in the *asynchronous* case requires some caution. Complete proofs for this case can be found in the third author forthcoming PhD thesis [10].

We introduce a notion of deterministic reduction $>_d$ which will be used to handle communications on return channels which are used at most once (see also remark 11(2)). Namely, $P >_d P'$ if

$$P \cong E[(\nu r)\,(\overline{r}(\vec{b}) \mid r(\vec{c}){:}Q)] \;\xrightarrow{\tau}\; E[(\mathbf{0} \mid ([\vec{b}/\vec{c}]Q \mid T_r))] \cong P' \cong E[[\vec{b}/\vec{c}]Q]$$

where $r \notin fn(Q)$ and E is an evaluation context (cf. section 2.2). We also write $\geq_d$ and $<_d$, $\leq_d$ with obvious meanings. It is easily verified that $>_d$ commutes, modulo $\cong$, with any transition:

[4] We expect that one can complicate the encoding to make the return channel r linear.

Lemma 15 *If $P \xrightarrow{\alpha} P'$ and $P >_d P''$ then either $P' \cong P''$ or for some Q, $P' >_d Q$ and $P'' \xrightarrow{\alpha} Q$.*

Let $\equiv_{aci}$ be the congruence on processes induced by the axioms for associativity and commutativity of parallel composition and $P \mid \mathbf{0} \equiv_{aci} P$. We define the following operators on relations:

$$\begin{array}{rcl} E(\mathcal{R}) & = & \{(P,Q) \mid P \equiv_{aci} (P' \mid M),\ Q \equiv_{aci} (Q' \mid M),\ (P,Q) \in \mathcal{R} \text{ and } M = \Pi_{i \in I}\overline{a_i}(\vec{b_i})\} \\ H_1(\mathcal{R}) & = & \sim \circ E(\mathcal{R}) \circ \sim \\ H_2(\mathcal{R}) & = & \geq_d \circ \mathcal{R} \circ \leq_d \\ H_3(\mathcal{R}) & = & \{((P \mid T_r),(Q \mid T_r)) \mid (P,Q) \in \mathcal{R}\} \end{array}$$

Lemma 16 (1) *The identity and the constant operators mapping a relation $\mathcal{R}$ to $\sim$, $\leq_d$, $\geq_d$, respectively, preserve $\leq_{\mathcal{F}}$.*
(2) *The operators E and H_i ($i = 1,2,3$) preserve $\leq_{\mathcal{F}}$.*

Proof. (1) the identity obviously preserves $\leq_{\mathcal{F}}$. For $\lambda\mathcal{R}. \sim$ we observe that a strong bisimulation is a weak bisimulation. For $\lambda\mathcal{R}. \leq_d$ and $\lambda\mathcal{R}. \geq_d$ we note that $\leq_d$ is a weak bisimulation.

(2) We note that all four operators are monotonic thus we just need to check that $\mathcal{R}_1 \leq_{\mathcal{F}} \mathcal{R}_2$ implies $H(\mathcal{R}_1) \subseteq \mathcal{F}(H(\mathcal{R}_2))$. □

Remark 17 *Unlike the synchronous case, the operator $H_1'(S) = \sim \circ S \circ \sim$ does not respect $\leq_{\mathcal{F}}$. For instance, consider, in a language extended with sum, $a.\overline{a} + \tau \sim \tau\, S\, \tau$ with $S = \{(\tau,\tau),(\mathbf{0},\mathbf{0})\}$. By computing first $E(S)$, we enforce closure under parallel composition with a message.*

We note the following property.

Lemma 18 *Suppose $I \vdash P$ and let S be an injective substitution on I. Then $S[\![P]\!] = [\![SP]\!]$ and $S\langle\!|I,P|\!\rangle = \langle\!|SI,SP|\!\rangle$.*

Then we can show that the encoding $\langle\!|I,P|\!\rangle$ 'simulates' P as follows:

Lemma 19 *Suppose $I \vdash P$. Then*
(1) *if $P \xrightarrow{a\vec{b}} P'$ then $\langle\!|I,P|\!\rangle \xrightarrow{a(j,\vec{b},r,c,c')} Q >_d \langle\!|I,P'|\!\rangle$,*
(2) *if $P \xrightarrow{(\nu\vec{w})\,\overline{a}\vec{b}} P'$, $I' \vdash P'$ and $a \notin I$ then $\langle\!|I,P|\!\rangle \xrightarrow{(\nu\vec{w},r)\,\overline{a}(\_,\vec{b},r,\_,\_)} (\langle\!|I',P'|\!\rangle \mid T_r)$,*
(3) *if $P \xrightarrow{\tau} P'$ then $\langle\!|I,P|\!\rangle \xrightarrow{\tau} Q >_d \langle\!|I,P'|\!\rangle$.*

Proof hint. We rely on the lemma 18, and we proceed by induction on the definition of the labelled transition relation. □

In the output case (2), we note the introduction of the T_r process acting on a fresh channel. We will show that we can factor out this spurious process. In particular, we observe the following property.

Lemma 20 *Suppose $r \notin fn(P \mid Q)$. Then $P \approx Q$ iff $(P \mid T_r) \approx (Q \mid T_r)$.*

Proof. ($\Rightarrow$) $\approx$ is preserved by parallel composition.

($\Leftarrow$) We show that the relation $\{(P,Q) \mid (P \mid T_r) \approx (Q \mid T_r) \text{ and } r \notin fn(P \mid Q)\}$ is a weak bisimulation up to injective substitution. □

Since the key in_a is kept restricted, a message $\overline{a}(j,\vec{b},r,c,c')$ received by a process $\langle\!|I,P|\!\rangle$ is always interpreted as an output request and therefore the fields j,r,c,c' are irrelevant. We formalize this remark as follows.

Lemma 21 *Let $I \vdash P$ and $a \in I$. Then for any j distinct from in_a, for any r,c,c'*

$$\langle\!|I,(P \mid \overline{a}(\vec{b}))|\!\rangle \sim (\langle\!|I,P|\!\rangle \mid \overline{a}(j,\vec{b},r,c,c'))$$

PROOF. Let $M = \Pi_{i\in I}\overline{a_i}(j_i, \vec{b}_i, r_i, c_i, c_i')$. We observe that

$$(CM(a, in_a) \mid \overline{a}(\_, \vec{b}, \_, \_, \_) \mid M) \sim (CM(a, in_a) \mid \overline{a}(j, \vec{b}, r, c, c') \mid M)$$

provided that $j \neq in_a$. Then we use the fact that bisimulation is preserved by evaluation contexts. $\Box$

Regarding the transitions of the encoding $\langle\!\langle I, P\rangle\!\rangle$, we have the following properties.

Lemma 22 *Suppose $I \vdash P$. Then:*

(1) *if $\langle\!\langle I, P\rangle\!\rangle \xrightarrow{\tau} Q$ then $P \xrightarrow{\tau} P'$ and $Q >_d \langle\!\langle I, P'\rangle\!\rangle$,*

(2) *if $\langle\!\langle I, P\rangle\!\rangle$ performs an output transition on a channel $a \notin I$ then $\langle\!\langle I, P\rangle\!\rangle \overset{(\nu\vec{w},r)\,\overline{a}(\_,\vec{b},r,\_,\_)}{\longrightarrow} (Q \mid T_r)$ and $P \overset{(\nu\vec{w})\,\overline{a}\vec{b}}{\longrightarrow} P'$ for some I' and P' such that $I' \vdash P'$ and $Q \cong \langle\!\langle I', P'\rangle\!\rangle$,*

(3) *if $\langle\!\langle I, P\rangle\!\rangle \overset{a(j,\vec{b},r,c,c')}{\longrightarrow} Q'$ then $P \overset{a\vec{b}}{\longrightarrow} P'$ for some P' such that $Q' >_d \langle\!\langle I, P'\rangle\!\rangle$.*

PROOF HINT. (1) the only receivers ready to execute in $\langle\!\langle I, P\rangle\!\rangle$ are the channel managers. Thus a τ transition corresponds to an application of the rule (CM_τ) synchronising an input and an output request.

(2) the only output actions arising from $\langle\!\langle I, P\rangle\!\rangle$ with subject $a \notin I$ are those produced by the encoding $[\![\overline{a}(\vec{b})]\!]$ of a message.

(3) this input transition corresponds to the execution of the rule (CM_{in}). $\Box$

Now we can prove the main result of this section, showing that our encoding into the calculus with constants $CM(a, in_a)$ is fully abstract with respect to weak bisimilarity.

Theorem 23 *Suppose $I \vdash P_1$ and $I \vdash P_2$ in the π_1-calculus. Then*

$$P_1 \approx P_2 \quad \textit{iff} \quad \langle\!\langle I, P_1\rangle\!\rangle \approx \langle\!\langle I, P_2\rangle\!\rangle$$

PROOF. ($\Rightarrow$) We define

$$\mathcal{S} = \{(\langle\!\langle I, P_1\rangle\!\rangle, \langle\!\langle I, P_2\rangle\!\rangle) \mid P_1 \approx P_2\}$$

We show in appendix B that $\mathcal{S}$ is a bisimulation up to $H_3 \circ H_2 \circ H_1$. It follows from proposition 3 that $\mathcal{S} \subseteq \approx$. Thus:

$$P_1 \approx P_2 \;\Rightarrow\; \langle\!\langle I, P_1\rangle\!\rangle \,\mathcal{S}\, \langle\!\langle I, P_2\rangle\!\rangle \;\Rightarrow\; \langle\!\langle I, P_1\rangle\!\rangle \approx \langle\!\langle I, P_2\rangle\!\rangle\ .$$

($\Leftarrow$) We define

$$\mathcal{S} = \{(P_1, P_2) \mid I \vdash P_1,\ I \vdash P_2 \text{ and } \langle\!\langle I, P_1\rangle\!\rangle \approx \langle\!\langle I, P_2\rangle\!\rangle\}$$

We show in appendix B that $\mathcal{S}$ is a bisimulation. Thus

$$\langle\!\langle I, P_1\rangle\!\rangle \approx \langle\!\langle I, P_2\rangle\!\rangle \;\Rightarrow\; P_1 \,\mathcal{S}\, P_2 \;\Rightarrow\; P_1 \approx P_2\ .\ \Box$$

4.3 Implementation of the channel manager

To conclude the proof of our result, it remains to show that the channel manager $CM(a, in_a)$ can be implemented adequately in π_1^r. In figure 3 we show such an implementation. To denote concisely the various points of the control, it is convenient to describe it as a system of recursive equations where m_k, for $k = 1, \ldots, 5$, stands for the vector $j_k, \vec{b}_k, r_k, c_k, c_k'$ which is assumed not to contain in_a. It is immediate to compile this system of recursive equations in our notation ($\mathsf{rec}\, A(\vec{b}).P$), that is, CM_1 really denotes the parametric recursive process

$$(\mathsf{rec}\, A_1(a, in_a).a(m_1).\, \underbrace{a(m_2).\mathsf{if}\ \cdots}_{CM_2(a,m_1)})$$

and similarly for $CM_i(a, m_1, m_2, c)$ (which are subterms of CM_1). One immediately verifies that $\{a\} \vdash^r CM_1(a, in_a)$, as expected.

Let us comment on this implementation. The channel manager first performs two inputs on a. The first input must be an output request and the second an input request (otherwise the process CM_1 loops back). Then the channel manager proceeds to make an internal choice. To this end, it generates two messages $\overline{a}(\_, \_, \_, c, c)$ and $\overline{a}(\_, \_, \_, c, c')$, of which one may be received by CM_3, and then either by CM_4 or CM_5. If

$$
\begin{array}{rcl}
CM_1(a, in_a) & = & a(m_1).CM_2(a, m_1) \\
CM_2(a, m_1) & = & a(m_2).[j_1 = in_a \vee j_2 \neq in_a](CM_1(a, in_a) \mid \overline{a}(m_1) \mid \overline{a}(m_2)), \\
& & \qquad (\nu c)\,(\overline{a}(\_,\_,\_,c,c) \mid (\nu c')\,\overline{a}(\_,\_,\_,c,c') \mid CM_3(a, m_1, m_2, c)) \\
CM_3(a, m_1, m_2, c) & = & a(m_3).[c_3 \neq c](CM_3(a, m_1, m_2, c) \mid \overline{a}(m_3)), \\
& & \qquad [c_3' = c]\,CM_4(a, m_1, m_2, c), \\
& & \qquad\quad CM_5(a, m_1, m_2, c) \\
CM_4(a, m_1, m_2, c) & = & a(m_4).[c_4 \neq c](CM_4(a, m_1, m_2, c) \mid \overline{a}(m_4)), \\
& & \qquad (CM_1(a, in_a) \mid \overline{a}(m_1) \mid \overline{a}(m_2)) \\
CM_5(a, m_1, m_2, c) & = & a(m_5).[c_5 \neq c](CM_5(a, m_1, m_2, c) \mid \overline{a}(m_5)), \\
& & \qquad (CM_1(a, in_a) \mid \overline{r_2}(\vec{b}_1))\ .
\end{array}
$$

Figure 3: Implementation of the channel manager

the message received in CM_3 is $\overline{a}(\_,\_,\_,c,c)$ the channel manager goes to state CM_4 and then loops back, otherwise it goes to state CM_5 and it enables a communication.

Let $[CM_1/CM]$ denote the operation of replacing the abstract channel manager CM by the implementation CM_1 described above (if CM were regarded as an identifier, this would just be the substitution $[CM_1/CM]$). Then for instance $[CM_1/CM][\![P]\!]$ and $[CM_1/CM]\langle\!| I, P |\!\rangle$ are now terms of the π_1^r. For the following proposition, we still consider terms of the calculus enriched with the constants $CM(a, in_a)$. The proof given in appendix C is a long but straightforward case analysis.

Proposition 24 (1) *Let* $M \cong \Pi_{i\in I}\overline{a}_i(j_i, \vec{b}_i, r_i, c_i, c_i')$. *Then*

$$(CM(a, in_a) \mid M) \approx (CM_1(a, in_a) \mid M)\ .$$

(2) *If* $I \vdash P$ *then* $\langle\!| I, P |\!\rangle \approx [CM_1/CM]\langle\!| I, P |\!\rangle$.

This concludes our argument. We have provided an encoding of π_1 into π_1^r which is fully abstract with respect to weak asynchronous bisimulation (with unique receiver). This notion of equivalence is relatively simple and well studied [4] but of course other equivalences could be considered such as *barbed congruence*. We also note that the encoding considered is not *uniform* in the sense of Palamidessi [12] and in particular it does not preserve 'distribution' because (i) it introduces a centralised coordinator (the channel managers) and (ii) it does introduce divergent behaviours because of the busy waiting phenomenon. Whether a uniform encoding exists remains to be seen.

References

[1] R. Amadio. An asynchronous model of locality, failure, and process mobility. In *Proc. Coordination 97, Springer Lect. Notes in Comp. Sci. 1282*, 1997. Extended version appeared as RR-INRIA 3109.

[2] R. Amadio. On modeling mobility. *Theoretical Computer Science*, 240:147–176, 2000.

[3] R. Amadio, G. Boudol, and C. Lhoussaine. The distributed receptive π-calculus. Technical report, INRIA Research Report 4080, November 2000.

[4] R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195:291–324, 1998.

[5] R. Amadio and C. Meyssonnier. On the decidability of fragments of the asynchronous π-calculus. In *Proc. EXPRESS01, Electronic Notes in Theoretical Computer Science*, volume 52.1, 2001. Also appeared as RR-INRIA 4241.

[6] G. Boudol. Typing the use of resources in a concurrent calculus. *Proc. ASIAN 97, Springer Lect. Notes in Comp. Sci. 1345*, 1997.

[7] C. Fournet and G. Gonthier. The reflexive CHAM and the join-calculus. *Proc. ACM Principles of Prog. Lang.*, 1996.

[8] U. Golz and A. Mycroft. On the relationship of CCS and Petri Nets. *Proc. ICALP84, Springer Lect. Notes in Comp. Sci. 172:196–208*, 1984.

[9] N. Kobayashi, B. Pierce, and D. Turner. Linearity in the π-calculus. *Proc. ACM Principles of Prog. Lang.*, 1996. Expanded version in ACM TOPLAS 21-5 (1999).

[10] C. Lhoussaine. *Types, réceptivité et mobilité.* PhD thesis, U. Aix-Marseille I, 2002 (Forthcoming).

[11] M. Merro and D. Sangiorgi. On asynchrony in name passing calculi. *Proc. ICALP98, Springer Lect. Notes in Comp. Sci. 1443*, 1998.

[12] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. *Proc. ACM Principles of Prog. Lang.*, 1997.

[13] C. Reutenauer. *Aspects mathématiques des réseaux de Petri.* Masson Editeur, 1988. Also available in english: The mathematics of Petri Nets, Prentice-Hall.

[14] D. Sangiorgi. The name discipline of uniform receptiveness. In *Proc. ICALP 97.* Springer Lect. Notes in Comp. Sci. 1256, 1997.

[15] D. Sangiorgi. On the bisimulation proof method. *Journal of Math. Structures in Comp. Sci.*, 8:447–479, 1998.

A Proof of proposition 7(2)

In the absence of name generation, the internal transitions operate over a finite set of names. Consequently, we can build a Petri Net that mimicks exactly the internal transitions of a given process P. Then checking the message deliverability property amounts to check in a finite number of cases variants of the liveness property.

We consider a typical situation. Suppose given a system of parameterless equations of the shape

$$A = a.(\Pi_{i\in i}\overline{a}_i \mid \Pi_{j\in j}A_j) \tag{4}$$

and an initial configuration $P = \Pi_{i\in I_o}\overline{a}_i \mid \Pi_{j\in J_o}A_j$. The operational semantics is the expected one: an internal transition is possible when a configuration contains a message $\overline{a}$ and an identifier A defined by $A = a \ldots$

Now we build a Petri Net that simulates exactly this system. We follow a rather standard notation: $t, \ldots$ stand for transitions, $p, \ldots$ for places, $m, \ldots$ for markings (*i.e.* vectors of natural numbers), $\rightarrow$ for the reduction relation on markings, and $Pre(p,t)$ for the number of directed edges from place p to transition t.

- Let N be the set of channel names $a, b, \ldots$ and I the set of process identifiers $A, B, \ldots$ in the system (there are finitely many). We take the set of places as the (disjoint) union of N and I. The intended interpretation is that a token at place a corresponds to a message $\overline{a}$ and a token at place A means that the control of a thread is at A. Following this interpretation we determine an initial marking m_o.
- To every equation of the shape (4) in the system we associate a transition t which is connected to places as follows: an edge from A to t, an edge from a to t, an edge from t to a_i for $i \in I$, and an edge from t to A_j for $j \in J$.

It is easy to generalise this construction to parametric systems of the shape $A(\vec{a}) = a(\vec{b}).(\Pi_{i\in i}\overline{a}_i\vec{a}_i \mid \Pi_{j\in j}A_j(\vec{a}_j))$. In particular, one replaces the input by an external sum; details can be found, *e.g.*, in [5].

Let us now turn to the message deliverability property at the level of Petri Nets. Fix a place $a \in N$ (here a token corresponds to a message).

- Let $T = \{t \mid Pre(a,t) > 0\}$ be the set of transitions that can consume a token (message) in place a.
- Let $Fire_T$ be the set of markings that can reach a marking where at least one transition $t \in T$ is *enabled* (*i.e.*, ready to fire).

• Let $M^a = \{m \mid m_a > 0\}$ where m_a is the a^{th} component of m. This is the set of marking having at least one token (message) at place a.

Message deliverability *with respect to place* a translates into the following condition: whenever $m_0 \xrightarrow{*} m \in M^a$ then $m \in \mathit{Fire}_T$.[5] This is equivalent to say that:

$$\mathit{Reach}(m_0) \cap (M^a \cap (\mathit{Fire}_T)^c) = \emptyset \tag{5}$$

where $\mathit{Reach}(m_0) = \{m \mid m_0 \xrightarrow{*} m\}$ is the set of markings reachable form m_0 and $\_^c$ is the set-theoretic complement.

We can then apply standard results in Petri Net theory (we refer in particular to [13, chapter 6]. First, it is easily checked that both Fire_T and M^a are *ideals*, *i.e.*, upper closed sets with respect to the pointwise order on markings. Every ideal is semi-linear and since semi-linear sets are closed under intersection and complementation, it follows that $M^a \cap (\mathit{Fire}_T)^c$ is a *semi-linear* set. In particular, it is the set of markings reachable from an initial marking of a Petri Net that can be effectively constructed. To conclude, apply the fact that the emptyness of the intersection of the markings reachable from two Petri Nets is decidable by reduction to the reachability problem for Petri Nets. $\square$

B Proof of theorem 23

($\Rightarrow$) We define

$$\mathcal{S} = \{(\langle\!|I, P_1|\!\rangle, \langle\!|I, P_2|\!\rangle) \mid P_1 \approx P_2\}$$

We show that $\mathcal{S}$ is a bisimulation up to $H_3 \circ H_2 \circ H_1$. We will only consider one half of the bisimulation condition, the other half follows by a symmetric argument, noting that the operators H_i preserve symmetry.

(τ) Suppose $\langle\!|I, P_1|\!\rangle \xrightarrow{\tau} Q_1$. Then:

$$\begin{array}{ll}
P_1 \xrightarrow{\tau} P_1' \text{ and } Q_1 >_d \langle\!|I, P_1'|\!\rangle & \text{by lemma 22(1),} \\
P_2 \stackrel{\tau}{\Rightarrow} P_2' \text{ and } P_1' \approx P_2' & \text{since } P_1 \approx P_2, \\
\langle\!|I, P_2|\!\rangle \stackrel{\tau}{\Rightarrow} \langle\!|I, P_2'|\!\rangle & \text{by lemma 19(1).}
\end{array}$$

Thus $Q_1 \geq_d \langle\!|I, P_1'|\!\rangle \; \mathcal{S} \; \langle\!|I, P_2'|\!\rangle$, as $P_1' \approx P_2'$.

(out) Suppose $\langle\!|I, P_1|\!\rangle \xrightarrow{(\nu\vec{w},r)\,\overline{a}(\_,\vec{b},r,\_,\_)} (\langle\!|I', P_1'|\!\rangle \mid T_r)$, according to lemma 22(2). Then:

$$\begin{array}{ll}
P_1 \xrightarrow{(\nu\vec{w})\,\overline{a}\vec{b}} P_1' & \text{and} \\
P_2 \stackrel{(\nu\vec{w})\,\overline{a}\vec{b}}{\Longrightarrow} P_2' \text{ and } P_1' \approx P_2' & \text{since } P_1 \approx P_2, \\
\langle\!|I, P_2|\!\rangle \stackrel{(\nu\vec{w},r)\,\overline{a}(_,\vec{b},r,_,_)}{\Longrightarrow} (\langle\!|I', P_2'|\!\rangle \mid T_r) & \text{by lemma 19.}
\end{array}$$

Thus $(\langle\!|I', P_1'|\!\rangle \mid T_r) \; H_3(\mathcal{S}) \; (\langle\!|I', P_2'|\!\rangle \mid T_r)$.

(in) Suppose $\langle\!|I, P_1|\!\rangle \xrightarrow{a(j,\vec{b},r,c,c')} Q_1$. Then

$$\begin{array}{ll}
P_1 \xrightarrow{a\vec{b}} P_1' \text{ and } Q_1 >_d \langle\!|I, P_1'|\!\rangle & \text{by lemma 22(3). Then either} \\
P_2 \stackrel{a\vec{b}}{\Rightarrow} P_2' \text{ and } P_1' \approx P_2' & \text{since } P_1 \approx P_2, \text{ hence} \\
\langle\!|I, P_2|\!\rangle \stackrel{a(j,\vec{b},r,c,c')}{\Longrightarrow} \langle\!|I, P_2'|\!\rangle & \text{by lemma 19, or} \\
P_2 \stackrel{\tau}{\Rightarrow} P_2' \text{ and } P_1' \approx (P_2' \mid \overline{a}(\vec{b})) & \text{by } P_1 \approx P_2, \\
\langle\!|I, P_2|\!\rangle \stackrel{\tau}{\Rightarrow} \langle\!|I, P_2'|\!\rangle & \text{by lemma 19(1).}
\end{array}$$

By lemma 21, $\langle\!|I, P_2' \mid \overline{a}(\vec{b})|\!\rangle \sim (\langle\!|I, P_2'|\!\rangle \mid \overline{a}(j, \vec{b}, r, c, c'))$. Thus:

$$Q_1 >_d \langle\!|I, P_1'|\!\rangle \; \mathcal{S} \; \langle\!|I, P_2' \mid \overline{a}(\vec{b})|\!\rangle \sim (\langle\!|I, P_2'|\!\rangle \mid \overline{a}(j, \vec{b}, r, c, c'))$$

[5] In our particular case, this implies that we can reach a marking containing a token in a place A such that the identifer A is defined by the equation $A = a \ldots$

($\Leftarrow$) We define

$$\mathcal{S} = \{(P_1, P_2) \mid I \vdash P_1,\ I \vdash P_2 \text{ and } \langle\!|I, P_1|\!\rangle \approx \langle\!|I, P_2|\!\rangle\}$$

We show that $\mathcal{S}$ is a bisimulation.
(τ) Suppose $P_1 \xrightarrow{\tau} P_1'$. Then

$$\begin{array}{ll}
\langle\!|I, P_1|\!\rangle \xrightarrow{\tau}>_d \langle\!|I, P_1'|\!\rangle & \text{by lemma 19(1),} \\
\langle\!|I, P_2|\!\rangle \stackrel{\tau}{\Rightarrow} Q_2 \text{ and } Q_2 \approx \langle\!|I, P_1'|\!\rangle & \text{since } \langle\!|I, P_1|\!\rangle \approx \langle\!|I, P_2|\!\rangle \\
P_2 \xrightarrow{\tau} P_2' \text{ and } Q_2 (>_d)^* \langle\!|I, P_2'|\!\rangle & \text{by lemmas 22(1) and 15.}
\end{array}$$

Thus $\langle\!|I, P_1'|\!\rangle \approx Q_2 \geq_d \langle\!|I, P_2'|\!\rangle$ which implies $P_1' \ \mathcal{S}\ P_2'$.

(out) Suppose $P_1 \stackrel{(\nu\vec{w})\overline{a}\vec{b}}{\rightarrow} P_1'$ with $a \notin I$. Then:

$$\begin{array}{ll}
\langle\!|I, P_1|\!\rangle \stackrel{(\nu\vec{w},r)\,\overline{a}(_,\vec{b},r,_,_)}{\rightarrow} (\langle\!|I', P_1'|\!\rangle \mid T_r) & \text{by lemma 19(2),} \\
\langle\!|I, P_2|\!\rangle \stackrel{(\nu\vec{w},r)\,\overline{a}(_,\vec{b},r,_,_)}{\Rightarrow} Q_2 \text{ and } (\langle\!|I', P_1'|\!\rangle \mid T_r) \approx Q_2 & \\
P_2 \stackrel{(\nu\vec{w})\overline{a}\vec{b}}{\rightarrow} P_2' \text{ and } Q_2 (>_d)^* (\langle\!|I', P_2'|\!\rangle \mid T_r) & \text{by lemmas } 22, 15, \\
 & \text{and diagram chasing.}
\end{array}$$

Thus $(\langle\!|I', P_1'|\!\rangle \mid T_r) \approx Q_2 (>_d)^* (\langle\!|I', P_2'|\!\rangle \mid T_r)$. By lemma 20, $\langle\!|I', P_1'|\!\rangle \approx \langle\!|I', P_2'|\!\rangle$, and therefore $P_1' \ \mathcal{S}\ P_2'$.

(in) Suppose $P_1 \xrightarrow{a\vec{b}} P_1'$. Then $\langle\!|I, P_1|\!\rangle \stackrel{a(j,\vec{b},r,c,c')}{\rightarrow} Q_1 >_d \langle\!|I, P_1'|\!\rangle$ by lemma 19(1). There are two cases: if

$$\begin{array}{ll}
\langle\!|I, P_2|\!\rangle \stackrel{a(j,\vec{b},r,c,c')}{\Rightarrow} Q_2 \text{ and } Q_1 \approx Q_2 & \text{by } \langle\!|I, P_1|\!\rangle \approx \langle\!|I, P_2|\!\rangle, \text{ then} \\
P_2 \stackrel{a\vec{b}}{\Rightarrow} P_2' \text{ and } Q_2 (>_d)^* \langle\!|I, P_2'|\!\rangle & \text{by lemmas 22 and 15.}
\end{array}$$

Thus from $\langle\!|I, P_1'|\!\rangle \approx Q_1 \approx Q_2 \approx \langle\!|I, P_2'|\!\rangle$, it follows $P_1' \ \mathcal{S}\ P_2'$. In the other case, if

$$\begin{array}{ll}
\langle\!|I, P_2|\!\rangle \stackrel{\tau}{\Rightarrow} Q_2 \text{ and } Q_1 \approx (Q_2 \mid \overline{a}(j, \vec{b}, r, c, c')) & \text{by } \langle\!|I, P_1|\!\rangle \approx \langle\!|I, P_2|\!\rangle, \text{ then} \\
P_2 \stackrel{\tau}{\Rightarrow} P_2' \text{ and } Q_2 (>_d)^* \langle\!|I, P_2'|\!\rangle & \text{by lemmas 22 and 15.}
\end{array}$$

By lemma 21, $\langle\!|I, P_2' \mid \overline{a}\vec{b}|\!\rangle \approx (\langle\!|I, P_2'|\!\rangle \mid \overline{a}(j, \vec{b}, r, c, c'))$. Thus from

$$\langle\!|I, P_1'|\!\rangle \approx Q_1 \approx (Q_2 \mid \overline{a}(j, \vec{b}, r, c, c')) \approx (\langle\!|I, P_2'|\!\rangle \mid \overline{a}(j, \vec{b}, r, c, c')) \approx \langle\!|I, P_2' \mid \overline{a}(\vec{b})|\!\rangle$$

it follows $P_1' \ \mathcal{S}\ (P_2' \mid \overline{a}(\vec{b}))$. $\square$

C Proof of proposition 24

(1) In this proof we use, in addition to $m_i = j_k, \vec{b}_k, r_k, c_k, c_k'$ as in the definition of CM_i (except that now we do not require that this vector does not contain in_a), the following notations:

$$\begin{array}{rclcrcl}
M & \cong & \Pi_{i \in I} \overline{a}(j_i, \vec{b}_i, r_i, c_i, c_i') & \quad & R & \cong & \Pi_{j \in J} \overline{r_j}(\vec{b}_j) \\
N & \cong & (M \mid R) & & m_{c_0,c_1} & = & \overline{a}(_, _, _, c_0, c_1)
\end{array}$$

Let $\mathcal{R}$ be the relation consisting of the following pairs:

$$\begin{array}{clr}
((CM(a, in_a) \mid N)\,,\ (CM_1(a, in_a) \mid N)) & & (1) \\
((CM(a, in_a) \mid \overline{a}(m_1) \mid N)\,,\ (CM_2(a, m_1) \mid N)) & & (2) \\
((CM(a, in_a) \mid \overline{a}(m_1) \mid \overline{a}(m_2) \mid N)\,,\ ((\nu c, c')\,(CM_3(a, m_1, m_2) \mid m_{c,c} \mid m_{c,c'}) \mid N)) & (*) & (3) \\
((CM(a, in_a) \mid \overline{a}(m_1) \mid \overline{a}(m_2) \mid N)\,,\ ((\nu c, c')\,(CM_4(a, m_1, m_2) \mid m_{c,c'}) \mid N)) & (*) & (4) \\
((CM(a, in_a) \mid \overline{r}_2\vec{b}_1 \mid N)\,,\ ((\nu c, c')\,(CM_5(a, m_1, m_2) \mid m_{c,c}) \mid N)) & & (5)
\end{array}$$

($*$) where $j_1 \neq in_a$ and $j_2 = in_a$. Then one checks that $\mathcal{R}$ is a weak bisimulation. Let $(P, Q) \in \mathcal{R}$ and $P \xrightarrow{\alpha} P'$. For each of the cases (1-5), we argue that we can find a matching transition $Q \stackrel{\alpha'}{\Rightarrow} Q'$ and still fall in one of the cases (1-5). In the following, we will describe schematically the matching transition by Q, omitting in particular the parameters of the CM_i's. For instance,

$$CM_4 \xrightarrow{\tau} CM_1 \xrightarrow{a} CM_2 \xrightarrow{\tau} CM_3$$

means that Q performs an input action on a, with appropriate arguments, preceeded and followed by internal synchronizations, and that following these transitions, the channel manager will move from state 4 to state 3 going through states 1 and 2. Whenever the sequence terminates in the state i, the reader should be able to verify that we fall in the i^{th} schema in the definition of the relation $\mathcal{R}$.

• If α is an output transition then it is caused by a message in R (its subject cannot be a, since a is in the interface of P). In all cases (1-5), the same message entails a matching transition by Q.

• Let us assume P's transition is obtained by means of rule CM_τ. We examine the five possible cases.

(1) To fire the transition, M must contain $\overline{a}(j_1, \vec{b}_1, r_1, c_1, c_1')$ and $\overline{a}(in_a, \vec{b}_2, r_2, c_2, c_2')$ with $j_1 \neq in_a$. Then Q matches P's transition with

$$CM_1 \xrightarrow{\tau} CM_2 \xrightarrow{\tau} CM_3 \xrightarrow{\tau} CM_5 \xrightarrow{\tau} CM_1$$

(2) M must contain at least $\overline{a}(j_2, \vec{b}_2, r_2, c_2, c_2')$. Q matches with

$$CM_2 \xrightarrow{\tau} CM_3 \xrightarrow{\tau} CM_4 \xrightarrow{\tau} CM_1 \xrightarrow{\tau} CM_2 \xrightarrow{\tau} CM_3 \xrightarrow{\tau} CM_5 \xrightarrow{\tau} CM_1$$

(3) Q matches with

$$CM_3 \xrightarrow{\tau} CM_4 \xrightarrow{\tau} CM_1 \xrightarrow{\tau} CM_2 \xrightarrow{\tau} CM_3 \xrightarrow{\tau} CM_5 \xrightarrow{\tau} CM_1$$

(4) Q matches with

$$CM_4 \xrightarrow{\tau} CM_1 \xrightarrow{\tau} CM_2 \xrightarrow{\tau} CM_3 \xrightarrow{\tau} CM_5 \xrightarrow{\tau} CM_1$$

(5) Q matches with

$$CM_5 \xrightarrow{\tau} CM_1 \xrightarrow{\tau} CM_2 \xrightarrow{\tau} CM_3 \xrightarrow{\tau} CM_5 \xrightarrow{\tau} CM_1$$

• Next let us assume P's transition is obtained by means of rule CM_{in}.

(1) M must contain an input request $\overline{a}(in_a, \vec{b}_2, r_2, c_2, c_2')$ and $j_1 \neq in_a$. Q matches with

$$CM_1 \xrightarrow{a} CM_2 \xrightarrow{\tau} CM_3 \xrightarrow{\tau} CM_5 \xrightarrow{\tau} CM_1$$

(2) We distinguish two cases.

(2.1) The transition consumes a message $\overline{a}(in_a, j, \vec{b}, r, c, c') \in M$. Q matches with

$$CM_2 \xrightarrow{\tau} CM_3 \xrightarrow{\tau} CM_4 \xrightarrow{\tau} CM_1 \xrightarrow{a} CM_2 \xrightarrow{\tau} CM_3 \xrightarrow{\tau} CM_5 \xrightarrow{\tau} CM_1$$

(2.2) The transition consumes the message $\overline{a}(m_1)$ with $j_1 = in_a$. Q matches with

$$CM_2 \xrightarrow{a} CM_1 \xrightarrow{\tau} CM_2 \xrightarrow{\tau} CM_3 \xrightarrow{\tau} CM_5 \xrightarrow{\tau} CM_1$$

(3) Q matches with

$$CM_3 \xrightarrow{\tau} CM_4 \xrightarrow{\tau} CM_1 \xrightarrow{a} CM_2 \xrightarrow{\tau} CM_3 \xrightarrow{\tau} CM_5 \xrightarrow{\tau} CM_1$$

(4) Q matches with

$$CM_4 \xrightarrow{\tau} CM_1 \xrightarrow{a} CM_2 \xrightarrow{\tau} CM_3 \xrightarrow{\tau} CM_5 \xrightarrow{\tau} CM_1$$

(5) Q matches with

$$CM_5 \xrightarrow{\tau} CM_1 \xrightarrow{a} CM_2 \xrightarrow{\tau} CM_3 \xrightarrow{\tau} CM_5 \xrightarrow{\tau} CM_1$$

• In the other direction, suppose $(P, Q) \in \mathcal{R}$ and $Q \xrightarrow{\alpha} Q'$. We proceed by a case analysis, as above.

(1) If CM_1 goes to CM_2 by an input or a synchronization then CM does nothing, i.e., $P \overset{\tau}{\Rightarrow} P$ with 0 transitions, and we fall in case (2).

(2) If CM_2 goes to CM_3 by an input or a synchronization then CM does nothing and we fall in case (3). On the other hand, if CM_2 goes to CM_1 by an input or a synchronization then CM does nothing and we fall in case (1).

(3) If CM_3 loops on itself by an input or a synchronization then CM does nothing and we stay in case (3). On the other hand, if CM_3 goes to CM_4 by a synchronization then CM does nothing and we fall in case (4).

Finally, if CM_3 goes to CM_5 by a synchronization then CM performs the corresponding synchronization and we fall in case (5).

(4) If CM_4 loops on itself by an input or a synchronization then CM does nothing and we stay in case (4). On the other hand, if CM_4 goes to CM_1 by a synchronization then CM does nothing and we fall in case (1).

(5) If CM_5 loops on itself by an input or a synchronization then CM does nothing and we stay in case (5). On the other hand, if CM_5 goes to CM_1 by a synchronization then CM does nothing and we fall in case (1).

(2) In the encoding, a channel manager $CM(a, in_a)$ can be in two positions:

(a) Under an input prefix, in a process of the shape:

$$(\nu a)\,(\nu in_a)\,(CM(a, in_a) \mid [\![P]\!])$$

We note that as long as $CM(a, in_a)$ (or $CM_1(a, in_a)$) is under the input prefix it does not play a role in the transitions and it is not affected by them because of the restrictions acting on its parameters.

(b) In a well-formed process that, up to structural equivalence, has the shape:

$$(\nu in_a)\,(CM(a, in_a) \mid P) \quad \text{or} \quad (\nu in_a)\,(\nu a)\,(CM(a, in_a) \mid P)$$

where P may contain the key in_a only in messages $\overline{a}(in_a, \vec{b}, r, c, c')$. Now in this case, one can show, by a case analysis similar to that in part (i), that the behaviour of $CM(a, in_a)$ placed in an evaluation context can be bisimulated by a suitable state $CM_i(a, in_a)$, $i = 1, \ldots, 5$ placed in the same evaluation context. $\square$