Asynchrony and the π-calculus

(Note)

Gérard Boudol

INRIA Sophia-Antipolis

06560-VALBONNE FRANCE

Abstract

We introduce an asynchronous version of Milner's π-calculus, based on the idea that the messages are elementary processes that can be sent without any sequencing constraint. We show that this simple message passing discipline, together with the restriction construct making a name private for an agent, is enough to encode the synchronous communication of the π-calculus.

1. Introduction.

The purpose of this note is to introduce an asynchronous variant of the π-calculus of Milner, Parrow and Walker [8]. The π-calculus is an extension of CCS, based on previous work by Engberg and Nielsen [5], that deals with *name passing*: in this calculus, agents pass channel names to other agents through named channels. The expressiveness of this link passing discipline has been demonstrated in [8] by a series of examples. Later on, Milner showed in [9] that even a restricted fragment of the original π-calculus is enough to encode the λ-calculus. More precisely, he showed that one can mimic in a "mini" π-calculus two reduction strategies for λ-terms, namely the lazy evaluation of Abramsky [1] and a weak version of the call-by-value evaluation of Plotkin [11]. This establishes that with the link passing discipline one has, to some extent, the power of passing agents as values.

It will be shown here that the "mini" π-calculus can be further reduced without affecting its expressive power. To introduce our version, let us recall some of the basic features of the original π-calculus, using the notations of [9] and [10]. To build π-agents, one presupposes an infinite set $\mathcal{N}$ of names, ranged over by $x, y, z, \ldots$ The most elementary agents are the *guarded* processes $g.P$, where the guard g may be of one of two kinds:

(i) an input guard $x(y)$, in which case the guarded agent $x(y).P$ waits for a name to be transmitted along the link named x. In this construct the name y is *bound*, and the agent $x(y).P$ can receive

This work has been partly supported by the Indo-French Centre for the Promotion of Advanced Research, Project 502-1.

any name z, to be substituted for y in P, yielding $P[z/y]$.

(ii) an output guard $\bar{x}z$, in which case the agent $\bar{x}z.P$ sends the name z along the link x, and then triggers P.

A communication may occur between two concurrent processes when one sends a name on a particular link and the other is waiting for a name along the *same* link. Denoting by $(P \mid Q)$ the parallel composition of two agents, a communication is the interaction between a sender $\bar{x}z.S$ and a receiver $x(y).R$, represented by the transition:

$$(\bar{x}z.S \mid x(y).R) \rightarrow (S \mid R[z/y])$$

This is typical of a *synchronous* message passing discipline, where the send operation is blocking: an output guard cannot be passed without the simultaneous occurrence of an input action.

One may argue that synchronous message passing is not very realistic as a primitive. Indeed, it is quite well-known that synchronous communication can be "implemented" in the more basic *asynchronous* message passing discipline, by means of a "mutual inclusion" protocol where the sender waits for an acknowledgement. Moreover, synchronous communication has still the flavour of passing an agent in a communication: the sender $\bar{x}z.S$ not only transmits the name z, but also gives an implicit *continuation* to the subprocess S. Using asynchronous communication, this continuation should be passed explicitly. This is exactly what we shall do here.

To explain the modification we shall propose on the π-calculus, let us return to the three characteristic features of synchronous message passing. The first one is that the input prefix is a (blocking) guard. This is a sensible assumption (typical of a weak, or lazy strategy of evaluation), since the future behaviour of the agent R in $x(y).R$ may depend on the actual value received for y. For instance if R is $\bar{y}v.R'$ or $y(u).R'$ then, depending on the link name received for y, this process will be able to communicate with different other agents. The second feature of synchronous communication is that a communication is an instantaneous interaction, that is a rendez-vous between several partners. Again this is a reasonable assumption, even in an asynchronous setting: for instance reading the value of a shared variable or getting the first available item from a buffer involves the participation of a receiving process *and* of a communication medium. The last feature of synchronous communication is that the sending operation is also a blocking guard. This means that the sending process is not allowed to continue before its message has been received, by a possibly distant agent. We shall not stay with this assumption here: output guards will be derived from lower-level constructs. This is the only modification we shall make on the π-calculus.

Usually, in asynchronous parallel programming languages, the messages are handled by means of communication structures, like buffers or mailboxes, in which they are stored. In the asynchronous π-calculus we propose, the communication media are just the (channel) names. However, we shall not give any particular status to these communication media. Instead, we shall regard the messages themselves as elementary agents, freely available for other processes waiting for them. This fits in very well with the idea of a *chemical abstract machine* of [2] for describing the evaluation process in the π-calculus. In this model, the state of a system of concurrent agents is viewed as a "chemical solution" (formally: a multiset of agents) in which floating "molecules" can interact with each other according to "reaction rules". In this setting, a message, that is a name z sent along a link x, will be just a particular molecule $\bar{x}z$ floating in the solution. This message can be received, or more precisely consumed by any agent $x(y).R$ floating in the same solution. The result of this interaction is a solution containing $R[z/y]$ in place of the message $\bar{x}z$ and the receiver $x(y).R$.

Formally, all this amounts to restrict the formation of a term $\bar{x}z.P$ in the π-calculus to the case where P is the "nil" process. But in fact we no longer need the "nil" agent, nor the guard notation $g.P$, since we shall not use any notion of "action". The syntax of our (mini) *asynchronous π-calculus* is thus:

$$P \ ::= \ \bar{x}z \ \mid \ x(y)P \ \mid \ (P \mid P) \ \mid \ !P \ \mid \ (\nu x)P$$

where x, y, z stand for names of $\mathcal{N}$, $!P$ is the *replication* construct of [9] and $(\nu x)P$ is the notation of [10] for the *restriction* (or *scoping*) construct, denoted $(x)P$ in [8,9]. In the next section of this note we shall develop the "chemical" semantics of this calculus. Then we will show how to encode Milner's (mini) π-calculus in our asynchronous calculus, using a simple protocol to achieve synchronous communication.

Related Work.

In the paper [7] Honda and Tokoro present a calculus which is nearly the same as the one introduced here. They also indicate how to encode synchronous communication in the asynchronous π-calculus, in a more complicated way than I will propose here, however. Moreover, the semantical perspective adopted here is different from the one of [7]: instead of bisimulations, we shall use Morris' extensional operational preorder. This involves defining a notion of observability of closed terms of the calculus, which is a direct adaptation of the observability of λ-terms. We shall prove here that the translation we propose from the synchronous π-calculus to the asynchronous one is adequate with respect to this semantics.

2. The asynchronous π-calculus.

In this section we introduce, using a chemical abstract machine, a notion of *reduction* of processes of our calculus. To this end, we first have to say a few words about substitution. As we said, the input prefix $x(y)$ acts as a *binder* for the name y. Similarly, the name x is bound in the term $(\nu x)P$. Then, as usual, substituting a name y for a name x in a term P, yielding $P[y/x]$, may require to rename some bound names to avoid binding y. We shall use $\mathsf{fn}(P)$ to denote the set of free names of the term P; we do not give the formal definition, which is obvious.

Let us now recall briefly the basic concepts of the chemical abstract machine (or CHAM); for more details, the reader is referred to [2]. The state $\mathcal{S}$ of a chemical machine is a "solution", that is a finite multiset of molecules. We shall use m, m' ... to range over molecules, and we denote by $\{\!|m_1, \ldots, m_k|\!\}$ finite multisets of molecules. The union of two multisets is denoted $\mathcal{S} \uplus \mathcal{S}'$. In any CHAM, a solution, encapsulated within a "membrane", can itself be considered as a single molecule. However, we shall not use any specific symbol to denote the membrane: in fact it will be simply represented by the brackets $\{\!|$ and $|\!\}$ enclosing a multiset of molecules. In the CHAM that we are defining for our asynchronous π-calculus, the molecules are given by:

(i) any term P of the calculus is a molecule;

(ii) any solution $\mathcal{S} = \{\!|m_1, \ldots, m_k|\!\}$ is a molecule;

(iii) if m is a molecule and x a name then $(\nu x)m$ is a molecule.

For instance if P and Q are terms of the calculus, $(\nu x)(\nu y)\{\!|\, P\,,\, Q\,|\!\}$ is a molecule. We shall use a context notation $m[\mathcal{S}]$ to denote a molecule containing a sub-molecule $\mathcal{S}$ which is a solution.

The behaviour of a CHAM consists in state changes, described as transitions $\mathcal{S} \Rightarrow \mathcal{S}'$. These transitions are given by means of axioms and rules. The axioms are specific to each CHAM describing the operational semantics of a particular language. They are of the form:

$$\{\!|\, m_1, \ldots, m_n \,|\!\} \Rightarrow \{\!|\, m'_1, \ldots, m'_k \,|\!\}$$

Such an axiom means that if the molecules $m_1, \ldots, m_n$ are floating in a solution $\mathcal{S}$ then a transformation of this solution may occur, resulting in a new solution $\mathcal{S}'$ where these molecules have been replaced by $m'_1, \ldots, m'_k$. This is formalized by stating that for any CHAM the following inference rule, called the *chemical law*, holds:

$$\boxed{\dfrac{\mathcal{S} \Rightarrow \mathcal{S}'}{\mathcal{S} \uplus \mathcal{S}'' \Rightarrow \mathcal{S}' \uplus \mathcal{S}''} \qquad (\textit{chemical law})}$$

There is another general law, called the *membrane law*, which asserts that transformations of solutions may occur within molecules:

$$\boxed{\dfrac{\mathcal{S} \Rightarrow \mathcal{S}'}{\{\!|\, m[\mathcal{S}] \,|\!\} \Rightarrow \{\!|\, m[\mathcal{S}'] \,|\!\}} \qquad (\textit{membrane law})}$$

These are the only two inference rules we shall use here. Regarding the axioms for transforming solutions, we distinguish three different kinds. The first kind of axioms is that of *heating* transformations. They usually take the form $\{\!|\, m \,|\!\} \Rightarrow \{\!|\, m'_1, \ldots, m'_k \,|\!\}$, and break a complex molecule into simpler ones. For these transformations we use the symbol $\rightharpoonup$ instead of $\Rightarrow$. Conversely, simple molecules can cool down to complex ones. These *cooling* transformations usually take the form $\{\!|\, m_1, \ldots, m_n \,|\!\} \Rightarrow \{\!|\, m' \,|\!\}$, and we shall use the symbol $\rightharpoondown$ for them. Most often, these transformations are reversible, and in this case we will denote by

$$\{\!|\, m_1, \ldots, m_n \,|\!\} \rightleftharpoons \{\!|\, m'_1, \ldots, m'_k \,|\!\}$$

a pair of inverse heating and cooling transformations (where, in general, $n = 1$). Finally the third kind of specific transformations is the one of *reactions*. Usually these transformations are irreversible; we shall use the symbol $\mapsto$ for them. By abuse of notation, we shall also use the symbols $\rightharpoonup$, $\rightharpoondown$ and $\mapsto$ for state changes, so that $\Rightarrow$ is the union of these three relations. As usual $\overset{*}{\rightleftharpoons}$ denotes the reflexive and transitive closure of $\rightleftharpoons$. From now on, we shall systematically forget about the outermost brackets $\{\!|$ and $|\!\}$ enclosing a solution.

Now we describe the specific transformations of the CHAM for our asynchronous π-calculus. The presentation that follows is a slight variation of the one given in [2] for the (mini) π-calculus of Milner [9]. There is only one reaction rule, formalizing the consumption of an available message by a receiver:

$$x(y)P\,,\, \bar{x}z \mapsto P[z/y] \qquad (\textit{reaction})$$

All the other axioms for the π-calculus are reversible heating and cooling transformations, intended to prepare communications, or conversely to restore an agent. The first transformation consists in breaking a compound agent into parts, which are then allowed to freely "move" in the solution:

$$(P \mid Q) \rightleftharpoons P\,,\, Q \qquad (\textit{parallel})$$

Similarly, a replicated agent $!P$ releases copies of P in the solution:

$$!P \rightleftharpoons P\,,\, !P \qquad (\textit{replication})$$

For instance, if we consider a "sender", that is a process of the form $(\bar{x}z \mid S)$, we can see that by heating a solution containing it, we make the message $\bar{x}z$ available for other molecules in the solution:

$$\ldots, (\bar{x}z \mid S), \ldots \rightleftharpoons \ldots, \bar{x}z, \ldots, S, \ldots$$

The remaining rules deal with the scoping construct $(\nu x)P$. Intuitively, the binder (νx) restricts the scope of the name x, by making it *private* to the agent P. In particular, the process $(\nu x)P$ is only allowed to communicate with its environment on channels different from x. However, internal communications may occur within the restricted agent. This is achieved in the CHAM by opening a new sub-solution for the restricted agent:

$$(\nu x)P \rightleftharpoons (\nu x)\{\!|\, P \,|\!\} \qquad \textit{(scoping membrane)}$$

For instance, applying the previous transformation rules we have the following transitions:

$$\begin{aligned}(\nu x)\big(x(y)R \mid \bar{x}z\big) &\rightleftharpoons (\nu x)\{\!|\, \big(x(y)R \mid \bar{x}z\big) \,|\!\}\\ &\rightleftharpoons (\nu x)\{\!|\, x(y)R\,,\, \bar{x}z \,|\!\}\\ &\mapsto (\nu x)\{\!|\, R[z/y] \,|\!\}\\ &\rightleftharpoons (\nu x)R[z/y]\end{aligned}$$

To allow a restricted process $(\nu x)P$ to communicate externally on channels other than x, there is a transformation called *scope migration* (*cf.* [8]):

$$((\nu x)P \mid Q) \rightleftharpoons (\nu x)(P \mid Q) \qquad (x \text{ not free in } Q) \qquad \textit{(scope migration)}$$

A typical use of this transformation is to prepare the sending of a private name, for instance by an agent like $S = (\nu z)(\bar{x}z \mid S')$, to another process which then falls in the scope of this name. Not only the receiving process acquires the name, but it also acquires the exclusive right of communicating on this channel with the sender. For instance, we have:

$$\begin{aligned}\big((P \mid x(y)R) \mid S\big) &\overset{*}{\rightleftharpoons} P\,,\, (x(y)R \mid S)\\ &\rightleftharpoons P\,,\, (\nu z)\big(x(y)R \mid (\bar{x}z \mid S')\big) \qquad (z \notin \mathsf{fn}(R))\\ &\overset{*}{\rightleftharpoons} P\,,\, (\nu z)\{\!|\, x(y)R\,,\, \bar{x}z\,,\, S' \,|\!\}\\ &\mapsto P\,,\, (\nu z)\{\!|\, R[z/y]\,,\, S' \,|\!\}\\ &\overset{*}{\rightleftharpoons} \big(P \mid (\nu z)(R[z/y] \mid S')\big)\end{aligned}$$

In general, in order to apply the previous transformation, one needs to make the private name different from any other name occurring in the environment. Then our last tranformation rule is:

$$(\nu x)P \rightleftharpoons (\nu y)P[y/x] \qquad (y \text{ not free in } P) \qquad \textit{(name conversion)}$$

Summarizing, the specific transformation rules of the CHAM for the asynchronous π-calculus are displayed in the following table:

$x(y)P\,,\, \bar{x}z$	$\mapsto$	$P[z/y]$		*(reaction)*
$(P \mid Q)$	$\rightleftharpoons$	$P\,,\, Q$		*(parallel)*
$!P$	$\rightleftharpoons$	$P\,,\, !P$		*(replication)*
$(\nu x)P$	$\rightleftharpoons$	$(\nu x)\{\!\|\, P \,\|\!\}$		*(scoping membrane)*
$((\nu x)P \mid Q)$	$\rightleftharpoons$	$(\nu x)(P \mid Q)$	$(x$ not free in $Q)$	*(scope migration)*
$(\nu x)P$	$\rightleftharpoons$	$(\nu y)P[y/x]$	$(y$ not free in $P)$	*(name conversion)*

To execute a process P, we start from the solution $\mathcal{S}_0 = \{\!|\, P \,|\!\}$, and perform transitions. It is easy to see that for any solution $\mathcal{S}$ there exists a term Q such that $\mathcal{S} \overset{*}{\rightleftharpoons} \{\!|\, Q \,|\!\}$, therefore we shall also use the symbols $\rightharpoonup$, $\rightharpoondown$ and $\mapsto$ to denote transitions between terms. More precisely, we shall say that:

(i) Q and R are *structurally equivalent* whenever $Q \overset{*}{\rightleftharpoons} R$;

(ii) the term Q *reduces* to R, in notation $Q \triangleright R$, whenever $Q \overset{*}{\rightleftharpoons}\mapsto\overset{*}{\rightleftharpoons} R$.

Our structural equivalence is slightly less generous than the one used by Milner in [9]. To recover this equivalence we should at least add the following two axioms:

$$(\nu x)P \rightleftharpoons P \qquad (x \text{ not free in } P) \qquad\qquad (\textit{scope extinction})$$
$$(\nu x)(\nu y)P \rightleftharpoons (\nu y)(\nu x)P \qquad\qquad (\textit{scoping exchange})$$

We denote by $\equiv$ the resulting extended heating and cooling equivalence, although this equivalence still does not coincide with the one of [9], because it is not a congruence (in general, we do not have $P \equiv Q \;\Rightarrow\; x(y)P \equiv x(y)Q$ or $P \equiv Q \;\Rightarrow\; !P \equiv !Q$ for instance).

3. Encoding the synchronous π-calculus.

In this section we show how to encode the synchronous (mini) π-calculus of [9] in our calculus. The syntax of the synchronous calculus is given by the grammar:

$$P \;::=\; \mathbf{0} \;\mid\; \bar{x}z.P \;\mid\; x(y).P \;\mid\; (P \mid P) \;\mid\; !P \;\mid\; (\nu x)P$$

where x, y, z stand for names of $\mathcal{N}$, and $\mathbf{0}$ denotes the "nil" process. In this section we call respectively π_a-calculus and π_s-calculus the asynchronous and synchronous versions of the π-calculus. The CHAM for the π_s-calculus is almost the same as the one of the π_a-calculus. Apart from the fact that the terms involved in the molecules are π_s-terms instead of π_a-terms, the only change concerns the reaction rule. This rule for the π_s-calculus is, as indicated in the introduction:

$$\boxed{x(y).P\,,\, \bar{x}z.Q \mapsto P[z/y]\,,\, Q \qquad (\pi_s\textit{-reaction})}$$

One can immediately note that, since the "nil" process $\mathbf{0}$ is "inert" (there is no rule for it), one may regard the π_a-calculus as a sub-calculus of the π_s-calculus, where $\bar{x}z$ is an abbreviation for $\bar{x}z.\mathbf{0}$, and $x(y)P$ stands for $x(y).P$. Alternatively, we could have used the same syntax for both the π_a and π_s-calculi, defining the asynchronous operational semantics by adding the rule:

$$\bar{x}z.S \rightleftharpoons \bar{x}z\,,\, S$$

However, we wanted to emphasize the fact that the restricted π_a-calculus is enough to encode synchronous communication.

The translation we shall give from the π_s to the π_a-calculus will be shown to be *adequate*. To explain what this means, let us first give some general definitions. Any given language L induces a notion of *context*: an L-context is a term C written using the constructs of the language L, to which is added a special constant $[]$, the "hole". Putting a term P of L into the holes in a context C gives a term denoted $C[P]$. Note that in languages involving binders, like the λ-calculus or the π-calculi, some free names of P may be bound by the context. We shall say that the context C *closes* the term P if $C[P]$ is closed. Unlike in the λ-calculus, where contexts of the form $\lambda x_1 \cdots x_n.[\,]$ may be used to close terms whose free variables belong to $\{x_1, \ldots, x_n\}$, there seems to be no obvious canonical way of closing π-terms. Using contexts of the form $(\nu x_1) \cdots (\nu x_n)[\,]$ would be absurd, since this would in general build deadlocked terms, like $(\nu x)x(y)P$ or $(\nu x)\bar{x}z$.

In fact, we shall use a notion of closedness π-term which is *different* from "containing no free names". Instead, we shall say that a π-term P is *closed* whenever it does not contain free messages. This means that if P contains a subterm $\bar{x}z$ (in the π_a-calculus) or $\bar{x}z.S$ (in the π_s-calculus), then both the channel name x and the object name z must be under the scope of some binder. In other words, only the channel names x of the input guards $x(y)$ may occur free in a closed term. For instance $(\nu z)(x(y)\bar{y}z)$ is closed, and a canonical way of closing a term whose free names belong to $\{y_1, \ldots, y_n\}$ is to put it in a context $x_1(y_1) \cdots x_n(y_n)[\,]$.

Now contexts may be used for *testing* operationally the terms of the language. We shall say that a term P passes a test C successfully whenever the program $C[P]$ *converges*, in notation $C[P]\Downarrow$. In the λ-calculus, this simply means that $C[P]$ has a normal form, with respect to some reduction strategy (see for instance [1] and [4]). However we do not regard this as a convenient notion of convergence for the π-calculi: here, as in [4], our criterion for convergence, or *observability* (see [10]), is the ability of being ready for a communication, possibly after some reduction. Then for instance "deadlocked" terms, or more accurately inaccessible agents like $(\nu x)x(y)P$ or $(\nu x)\bar{x}z$ though irreducible, will not be regarded as convergent ([1]). Moreover, some terms which do not possess any normal form will be considered as observable. Indeed, we shall regard a closed term $(P \mid Q)$ as convergent if P is convergent. We shall define this notion only for closed terms, so that convergence is *the ability of accepting some input*, exactly like in the lazy λ-calculus of Abramsky [1] or in the calculi for parallel functions of [4]. We first define the immediate convergence of closed terms on a particular channel name x, in notation $P \downarrow x$, to be the least predicate satisfying:

(i) $x(y)P \downarrow x$

(ii) $P \downarrow x \;\Rightarrow\; \begin{cases} (P \mid Q) \downarrow x & \text{and} \\ (Q \mid P) \downarrow x & \text{and} \\ !P \downarrow x \end{cases}$

(iii) $P \downarrow x \;\&\; x \neq y \;\Rightarrow\; (\nu y)P \downarrow x.$

This definition holds for π_a-terms as well as for π_s-terms (introducing a dot in the input guard). One should remark that immediate convergence cannot be destroyed by reduction: if P is a closed term such that $P \downarrow x$ then $P \stackrel{*}{\rightleftharpoons} (\nu x_1) \cdots (\nu x_n)(x(y)R \mid Q)$ with $x \notin \{x_1, \ldots, x_n\}$ and, since P is closed, there is no free message $\bar{x}z$ in Q, therefore if $P \rhd^* P'$ we have $P' \stackrel{*}{\rightleftharpoons} (\nu x_1) \cdots (\nu x_n)(x(y)R \mid Q')$ with $Q \rhd^* Q'$. Then a closed term that is immediately observable is a kind of "partial normal form", as in [4]. Now the convergence of closed terms is defined by:

$$P\Downarrow \;\Leftrightarrow_{\mathrm{def}}\; \exists Q\; \exists x \in \mathcal{N}.\quad P \rhd^* Q \;\;\&\;\; Q \downarrow x$$

We shall use indexed symbols $\Downarrow_a$ and $\Downarrow_s$ for convergence in the π_a and π_s-calculi. It is easily checked that convergence is preserved by structural equivalence, that is

$$P \stackrel{*}{\rightleftharpoons} Q \;\&\; Q\Downarrow \;\Rightarrow\; P\Downarrow$$

([1]) one should note that our notion of convergence will therefore be different from the one used by Milner in [9].

One should also note that

$$P \equiv Q \ \&\ Q\Downarrow \ \Rightarrow\ P\Downarrow$$

The semantics we shall use here for any language L is the extensional operational preorder of Morris (see [1,4]), also called *testing preorder* (²), defined by:

$$P \sqsubseteq_L Q \ \Leftrightarrow_{\mathrm{def}}\ \forall C \text{ closing } P \text{ and } Q.\ \ C[P]\Downarrow \ \Rightarrow\ C[Q]\Downarrow$$

For the π_a and π_s-calculi we shall use respectively the symbols $\sqsubseteq_a$ and $\sqsubseteq_s$, and we denote by $\simeq_a$ and $\simeq_s$ the associated equivalences, which are obviously congruences. Finally we shall say that an interpretation $[\![.]\!]$ of the terms of L into some domain, pre-ordered by $\leq$, is *adequate* if and only if any operational discrimination that can be made is reflected in the interpretation, that is:

$$[\![P]\!] \leq [\![Q]\!] \ \Rightarrow\ P \sqsubseteq_L Q$$

Our aim now is to show that there is an adequate interpretation of the synchronous π-calculus into the asynchronous π-calculus, equipped with the testing preorder.

To this end, we have to simulate synchronous communication in the π_a-calculus. Let us consider an example: suppose that we wish to build a system behaving like the π_s-agent $(\bar{x}z.S \mid x(y).R)$. In the asynchronous calculus the sending agent would be written $(\bar{x}z \mid S)$, but we have to prevent the subprocess S from being active until the message $\bar{x}z$ has been actually received. Then an idea is to *guard* the agent S by the reception of an acknowledgement, that is an explicit continuation, writing the sender as:

$$S' = (\bar{x}z \mid u(v)S)$$

assuming that v is not free in S. Symmetrically, the receiver would send the acknowledgement just after having received z along x, that is:

$$R' = x(y)(\bar{u}v \mid R)$$

(provided that u is not free in R). Unfortunately, we cannot apply this simple transformation independently from the context, since in this synchronization protocol there is no particular relation linking the communication channel x with the synchronization channel u. This last name should be known only by the sender and the receiver, while here it can be used also by the environment to interrupt the communication between S' and R'. For instance S' may accept a message on u from the environment.

To achieve a non-interruptible synchronization, we shall use a more elaborate protocol, in which the sending and receiving agents first exchange *private* links before performing the actual communication. The key observation is that, due to the restriction, in a sender like

$$(\nu u)(\bar{x}u \mid u(v)S)$$

the subprocess $u(v)S$ cannot proceed unless the message $\bar{x}u$ has been received by some other agent. Moreover, the channel name u will serve as a link only between $u(v)S$ and the receiver of $\bar{x}u$. Using

(²) this should not be confused with the testing preorders of Hennessy [6] and Boreale & De Nicola [3], which are far more discriminating.

this remark, we may now define our translation from the π_s-calculus into the π_a-calculus:

$$
\begin{array}{|rcll|}
\hline
[\![\mathbf{0}]\!] &=& (\nu x)(\nu z)\bar{x}z & \\
[\![\bar{x}z.P]\!] &=& (\nu u)\big(\bar{x}u \mid u(v)(\bar{v}z \mid [\![P]\!])\big) & (u,\, v \text{ not free in } P) \\
[\![x(y).P]\!] &=& x(u)(\nu v)(\bar{u}v \mid v(y)[\![P]\!]) & (u,\, v \text{ not free in } P) \\
[\![P \mid Q]\!] &=& ([\![P]\!] \mid [\![Q]\!]) & \\
[\![!P]\!] &=& ![\![P]\!] & \\
[\![(\nu x)P]\!] &=& (\nu x)[\![P]\!] & \\
\hline
\end{array}
$$

Some comments are in order. Note that in the coding of the guarded terms $\bar{x}z.P$ and $x(y).P$ the channel x is first used to exchange a private name u, the meaning of which is that the receiver will be engaged in a rendez-vous with the sender. Then the receiver sends back a private acknowledgement v on u, confirming the rendez-vous. Only after these "formalities" the actual transmission of z may occur, on the name v which is only known by the sender and the receiver. One should also note that the translation respects the free names, that is $\mathsf{fn}([\![P]\!]) = \mathsf{fn}(P)$, and that $[\![P]\!]$ is closed if and only if P is closed. It should be clear that the following holds:

$$P \overset{*}{\rightleftharpoons} Q \;\Rightarrow\; [\![P]\!] \overset{*}{\rightleftharpoons} [\![Q]\!]$$

We are now ready to establish the announced result:

THEOREM. *The translation $[\![.]\!]$ from the π_s-calculus to the π_a-calculus is adequate, that is, for any π_s-terms P and Q:*

$$[\![P]\!] \sqsubseteq_a [\![Q]\!] \;\Rightarrow\; P \sqsubseteq_s Q$$

PROOF SKETCH. We first show the computational adequacy of the interpretation, that is:

$$\forall P \text{ closed} \quad P{\Downarrow_s} \;\Leftrightarrow\; [\![P]\!]{\Downarrow_a} \tag{1}$$

Suppose that $[\![P]\!]{\Downarrow_a}$, that is there exist $w \in \mathcal{N}$ and Q such that $Q \downarrow w$ and $[\![P]\!] \triangleright^* Q$. We proceed by induction on the length of this reduction, showing that there exists Q' such that $P \triangleright^* Q'$ and $Q' \downarrow w$. If this length is 0, that is $[\![P]\!] \overset{*}{\rightleftharpoons} Q$, then we have $[\![P]\!] \downarrow w$, and it is easy to see that this is equivalent to $P \downarrow w$. Otherwise, since there is a reaction occurring in $[\![P]\!]$, one may check that there exist R, S and T such that

$$[\![P]\!] \overset{*}{\rightleftharpoons} (\nu x_1)\cdots(\nu x_n)\big(([\![x(y).R]\!] \mid [\![\bar{x}z.S]\!]) \mid [\![T]\!]\big)$$

and the first reaction occurring in the reduction $[\![P]\!] \triangleright^* Q$ is an interaction between $[\![x(y).R]\!]$ and $[\![\bar{x}z.S]\!]$. Then if we let $R' = (\nu v)(\bar{u}v \mid v(y)[\![R]\!])$ and $S' = u(v)(\bar{v}z \mid [\![S]\!])$, that is $[\![x(y).R]\!] = x(u)R'$ and $[\![\bar{x}z.S]\!] = (\nu u)(\bar{x}u \mid S')$, the first interaction between $[\![x(y).R]\!]$ and $[\![\bar{x}z.S]\!]$ is:

$$
\begin{array}{rcll}
([\![x(y).R]\!] \mid [\![\bar{x}z.S]\!]) & \overset{*}{\rightleftharpoons} & (\nu u)\big(x(u)R' \mid (\bar{x}u \mid S')\big) & (u \text{ not free in } R) \\
& \triangleright & (\nu u)(R' \mid S') &
\end{array}
$$

with

$$(\nu x_1)\cdots(\nu x_n)\big(\,(\nu u)(R' \mid S')\,\mid\,[\![T]\!]\big)\;\rhd^*\;Q$$

If this reduction sequence consists in reactions occurring only within $[\![T]\!]$ then

$$Q \overset{*}{\rightleftharpoons} (\nu x_1)\cdots(\nu x_n)\big(\,(\nu u)(R' \mid S')\,\mid\,T'\big)$$

with $[\![T]\!] \rhd^* T'$ and $T' \downarrow w$ with $w \neq x_i$ for any i, therefore by induction hypothesis there exists T'' such that $T \rhd^* T''$ and $T'' \downarrow w$, and we have

$$\begin{array}{rl} P & \overset{*}{\rightleftharpoons}\; (\nu x_1)\cdots(\nu x_n)\big((x(y).R \mid \bar{x}z.S) \mid T\big) \\ & \rhd^*\; (\nu x_1)\cdots(\nu x_n)\big((R[z/y] \mid S) \mid T''\big) \end{array}$$

Otherwise we have:

$$\begin{array}{rll} (\nu u)(R' \mid S') & \overset{*}{\rightleftharpoons}\; (\nu u)(\nu v)\big(\,(\bar{u}v \mid v(y)[\![R]\!])\,\mid\,u(v)(\bar{v}z \mid [\![S]\!])\,\big) & (v \text{ not free in } S) \\ & \rhd\; (\nu u)(\nu v)\big(\,v(y)[\![R]\!]\,\mid\,(\bar{v}z \mid [\![S]\!])\,\big) & \end{array}$$

with

$$(\nu x_1)\cdots(\nu x_n)\Big((\nu u)(\nu v)\big(\,v(y)[\![R]\!]\,\mid\,(\bar{v}z \mid [\![S]\!])\,\big)\,\mid\,[\![T]\!]\Big)\;\rhd^*\;Q$$

If this reduction sequence consists in reactions occurring within $([\![S]\!] \mid [\![T]\!])$ then we conclude as in the previous case. Otherwise we have:

$$\begin{array}{rll} (\nu u)(\nu v)\big(\,v(y)[\![R]\!]\,\mid\,(\bar{v}z \mid [\![S]\!])\,\big) & \rhd\; (\nu u)(\nu v)([\![R]\!][z/y] \mid [\![S]\!]) & \\ & \equiv\; ([\![R]\!][z/y] \mid [\![S]\!]) & (u,\, v \text{ not free in } R,\, S) \end{array}$$

with

$$(\nu x_1)\cdots(\nu x_n)\Big((\nu u)(\nu v)([\![R]\!][z/y] \mid [\![S]\!])\,\mid\,[\![T]\!]\Big)\;\rhd^*\;Q$$

Here again this implies that P converges on w.

The converse implication, that is $P{\Downarrow_s} \;\Rightarrow\; [\![P]\!]{\Downarrow_a}$ is shown in an entirely similar manner. This direction of the computational adequacy property is in fact easier to prove.

A second step in the proof of the theorem consists in showing that the translation is compatible with the constructs of the π_s-calculus, that is:

$$[\![P]\!] \sqsubseteq_a [\![Q]\!] \;\Rightarrow\; \forall C\ \pi_s\text{-context}\ \ [\![C[P]]\!] \sqsubseteq_a [\![C[Q]]\!] \tag{2}$$

To prove this point, we first remark that, since the translation $[\![.]\!]$ is defined in a compositional way, there exits a π_a-context C' such that $[\![C[R]]\!] = C'[\,[\![R]\!]\,]$ for any R. Now assume that for some π_a-context C'' we have $C''[\,[\![\,C[P]\,]\!]\,] = C''[\,C'[\,[\![P]\!]\,]\,]{\Downarrow}$. Then $C''[\,[\![\,C[Q]\,]\!]\,] = C''[\,C'[\,[\![Q]\!]\,]\,]{\Downarrow}$ follows from the hypothesis $[\![P]\!] \sqsubseteq_a [\![Q]\!]$.

Clearly, combining the points (1) and (2) above, we have shown the adequacy of the translation of the π_s-calculus into the π_a-calculus ❑

Acknowledgments The ideas of this note were worked out during a visit at the School of Mathematics, SPIC Science Foundation, Madras. I wish to thank P.S. Thiagarajan for the stimulating discussions we had, and for his hospitality. I am grateful to D. Sangiorgi for pointing out to me the paper by Honda and Tokoro.

REFERENCES

[1] S. Abramsky, *The lazy lambda-calculus*, in Research Topics in Functional Programming (D. Turner, Ed.), Addison-Wesley (1989) 65-116.

[2] G. Berry, G. Boudol, *The chemical abstract machine*, Theoretical Comput. Sci. 96 (1992) 217-248.

[3] M. Boreale, R. de Nicola, *Testing equivalence for mobile processes*, to appear in the Proceedings of the CONCUR Conference (1992).

[4] G. Boudol, *Lambda-calculi for (strict) parallel functions*, INRIA Res. Report 1387 (1991) to appear in Information and Computation.

[5] U. Engberg, M. Nielsen, *A calculus vvof communicating systems with label passing*, Daimi PB-208, Aarhus University (1986).

[6] M. Hennessy, *A model for the π-calculus*, Report 8/91, University of Sussex (1991).

[7] K. Honda, M. Tokoro, *An object calculus for asynchronous communication*, ECOOP 91, Lecture Notes in Comput. Sci. 512 (1991) 133-147.

[8] R. Milner, J. Parrow, D. Walker, *A calculus of mobile processes*, Information and Computation 100 (1992) 1-77.

[9] R. Milner, *Functions as processes*, Math. Struct. in Comp. Science 2 (1992) 119-141.

[10] R. Milner, *The polyadic π-calculus: a tutorial*, Technical Report ECS-LFCS-91-180, Edinburgh University (1991) Reprinted in *Logic and Algebra of Specification*, F. Bauer, W. Brauer and H. Schwichtenberg, Eds, Springer Verlag, 1993, 203-246.

[11] G. Plotkin, *Call-by-name, call-by-value and the λ-calculus*, Theoret. Comput. Sci. 1 (1975) 125-159.

[12] D. Sangiorgi, *The lazy lambda-calculus in a concurrency scenario*, LICS 92 (1992) 102-111.