Non-deterministic Lazy it λ-calculus VS it π-caculus

Carolina Lavatelli

Laboratoire d'Informatique, URA 1327 du CNRS
Département de Mathématiques et d'Informatique
Ecole Normale Supérieure
*RIMS Kyoto University

LIENS - 93 - 15

September 1993

Non-deterministic lazy λ-calculus vs π-calculus

Carolina Lavatelli

Laboratoire d'Informatique de l'Ecole Normale Supérieure
45, rue d'Ulm - 75230 Paris Cedex 05 - France
email:lavatel@dmi.ens.fr

September 1993

Abstract

We pursue the study of the embedding of the λ-calculus into the π-calculus. Various lambda calculi with parallel and convergence testing facilities are examined and their expressiveness compared; λ_j -a lazy calculus augmented with a non-deterministic choice operator and a convergence testing combinator, emerges as a suitable language to be encoded in π. Through the use of closures for variables and abstractions, the process of substitution in λ_j is managed in a semi-explicit way. The semantics associated to both λ_j and π are based on contextual testing preorders. We define an encoding of λ_j into π; we prove that it is adequate with respect to those semantics. However, the encoding is not fully-adequate; standard examples show that π is still more discriminating than λ_j.

1 Introduction.

Many attempts have been made in view of approaching the understanding of functions and concurrent processes, all of them issuing or originated in enrichements of the traditional support calculi, λ-calculus and CCS-like systems, which aim at breaking both the rigid structure of process algebra and the intrinsically sequential nature of λ-calculus. Striking examples are the γ-calculus [3, 2] and the Calculus of Higher-order Communicating Systems (CHOCS) [16], allowing any kind of entity to be passed in a communication, and the π-calculus of Milner, Parrow and Walker [10], based on the communication of access to processes rather than processes themselves. In relation with π, we can also mention its generalization to a higher-order calculus $HO\pi$ [17] and its asynchronous variants [6, 8]. One of the appealing features of those calculi is that they embed lazy and eager versions of pure λ-calculus, thus enabling a comparison of expressiveness.

In this paper we pursue the study of the embedding of the lazy λ-calculus into the π-calculus. Let us recall some terminology about Abramsky's lazy λ-calculus [1]. It consists of the following syntax for terms

$$M ::= x \mid \lambda x.M \mid (M\,M)$$

together with an evaluation mechanism, formalized by a convergence predicate $\Downarrow$, which allows to perform β-contractions of the form

$$(\lambda x.M)N \rightarrow M[N/x]$$

at the outermost level of terms or on subterms in function position. Values are closed abstractions; $M \Downarrow K$ means that the term M evaluates to K (or has the value K or converges to K); we write $M \Downarrow$ when the value is inessential and $M \Uparrow$ to denote divergence. Throughout the paper, I will stand for the identity function $\lambda x.x$. Terms are usually seen as operationally equivalent if

they belong to the equivalence $\simeq^B$ induced by the following applicative bisimulation preorder: For two closed λ-terms L and M,

$$(1) \qquad L \sqsubseteq^B M \quad \text{iff} \quad \text{for all sequences } \tilde{N} \text{ of closed terms } \ (L\tilde{N}) \Downarrow \Rightarrow \ (M\tilde{N}) \Downarrow \ .$$

On the other hand, π-calculus is an extension of CCS based upon the communication of channel names and the dynamic creation of fresh names. The standard notions of strong and weak bisimulation, denoted $\sim$ and $\sim^w$ respectively, are adapted.

Undoubtedly, one can say that the embedding of lazy λ-calculus into π casts some light on the descriptive power of this calculus. Milner [11] presents the embedding through a syntax-directed translation of λ-terms into π-processes, called $[\![\,]\!]$. The main result is an adequacy result involving $\sim^w$ and $\simeq^B$: For all closed λ-terms M and N,

$$(2) \qquad [\![M]\!] \sim^w [\![N]\!] \Rightarrow M \simeq^B N \ .$$

It is worth noticing that all the aforementioned embeddings enjoy an adequacy result of this kind. Furthermore, the converse of (2) is not true. That is, π-calculus is strictly more discriminating than lazy λ-calculus (a similar result holds when considering call-by-value λ-calculus). The sources of non-full-adequacy of the encoding are manifold; some of them must be found in a certain lack of expressiveness of the lazy λ-calculus; others follow from somehow inevitable design decisions taken in the definition of $[\![\,]\!]$. Examples of the first kind are built up with the aid of convergence and parallel convergence testing combinators $\mathbf{C}$ and $\mathbf{P}$ respectively. While they are non-definable in the lazy λ-calculus, there are processes that behave as them. Therefore, some well-known λ-terms (see Ong [14]) are applicative bisimilar while their corresponding processes are not weakly bisimilar.

One of the purposes of this paper is, in a first half, to describe and compare lambda calculi for which the first kind of counter-examples to full-adequacy are disallowed. That is, calculi with enough descriptive features to permit the definition of $\mathbf{C}$ and $\mathbf{P}$, so more suitable to be encoded into π than the pure λ-calculus. Recall that these combinators satisfy the following clauses:

$$\begin{cases} M \Downarrow \Rightarrow \ \mathbf{C}M \Downarrow I & \text{and} \quad M \Uparrow \Rightarrow \ \mathbf{C}M \Uparrow \\ (M \Downarrow \ \text{or} \ N \Downarrow) \ \Rightarrow \ \mathbf{P}MN \Downarrow I & \text{and} \quad (M \Uparrow \ \text{and} \ N \Uparrow) \ \Rightarrow \ \mathbf{P}MN \Uparrow \end{cases}$$

and observe that in the presence of $\mathbf{P}$, $\mathbf{C}$ is definable as $\lambda x.\mathbf{P}xx$. However, the calculus obtained by adding $\mathbf{P}$ to the lazy λ-calculus is not quite appealing in itself, at least from a programming language point of view.

Following the lines of the work on (strict) parallel functions by Boudol [4, 5], our approach consists in separating convergence testing from parallelism. We will show how the former facility can be modeled also by new forms of abstractions, namely value abstractions $\lambda^v x.M$ and strict abstractions $\lambda^s x.M$. Their meaning being the following: The evaluation of $(\lambda^v x.M)N$ asks for the convergence of N to a value K before contracting it to $M[K/x]$; that of $(\lambda^s x.M)N$ asks too for $N \Downarrow K$ but then evolves to $M[N/x]$. In fact, we prove that, in the presence of the lazy abstraction, $\mathbf{C}$, $\lambda^v x.M$ and $\lambda^s x.M$ are mutually definable. For example, $\lambda^v x.M = \lambda x.(Cx)M$ and $\mathbf{C} = \lambda^v x.I$.

As for the introduction of parallelism, two major apparently different options arise: one of them is to add a new operator $\|$ with interleaving semantics, the other is to add the non-deterministic internal choice $\oplus$. An evaluation of $M \,\|\, N$ consists in the concurrent evaluation of M and N. An evaluation of $M \oplus N$ is an evaluation of M or an evaluation of N.

The first point to stress is that from the point of view of convergence those parallel calculi are equivalent. Note that the combination of $\mathbf{C}$ with $\|$ or with $\oplus$ gives $\mathbf{P}$: It can be defined either as $\lambda x.\lambda y.(\mathbf{C}x \,\|\, \mathbf{C}y)$ or as $\lambda x.\lambda y.(\mathbf{C}x \oplus \mathbf{C}y)$. Moreover, the simultaneous addition of $\mathbf{C}$ and $\|$ or of $\mathbf{C}$ and $\oplus$ to the lazy λ-calculus is safe, i.e. the calculi have a context lemma. This is equally true if we take the strict abstraction instead of $\mathbf{C}$, but the symmetry is broken if we consider the value abstraction. Indeed, one cannot safely mix $\oplus$ with call-by-value applications. Therefore, all this extended λ-calculi -except the last one, are appropriate to be embedded into

π. For many technical reasons the best one is that calculus -called λ_j as in [4], which put together $\oplus$ and $\mathbf{C}$, besides lazy application.

A central issue (pointed out in [11]) is to find an extension of the λ-calculus as discriminating as the π-calculus, a step toward which is given by Sangiorgi in [18]. The approach consists in augmenting the lazy λ-calculus with a non-deterministic operator $\uplus$ such that $\uplus M$ acts either as M or as Ω (the paradigmatic divergent term $(\lambda x.xx)(\lambda x.xx)$), and in defining an extended notion of applicative bisimulation $\simeq^{\uplus}$ for this new calculus: $M \simeq^{\uplus} N$ iff there is a symmetric binary relation $\mathcal{R}$ on closed $\lambda_{\uplus}$-terms such that the two statements below hold:

- If $M \xrightarrow{\star} \lambda x.M'$ then $N \xrightarrow{\star} \lambda x.N'$ and $\forall$ closed $L \in \Lambda_{\uplus}$. $M'[R/x] \mathcal{R} N'[R/x]$
- If $M \xrightarrow{\star} M'$ then $N \xrightarrow{\star} N'$ and $M' \mathcal{R} N'$

The result is that, on closed pure λ-terms, $\lambda_{\uplus}$ has the same discriminatory power than π, i.e. for any $M, N \in \Lambda$ $[\![M]\!] \sim^{w} [\![N]\!]$ iff $M \simeq^{\uplus} N$.

It should be stressed that one cannot get a similar result if, instead of considering such modified version of applicative bisimulation, one keeps the standard contextual semantics. As we show in this paper, non-determinacy is not the only extra-feature of π with respect to λ if this calculus is provided with Morris's testing equivalence.

Let us be a little more precise. Contexts are built up as terms, possibly with a hole [] as a subterm; $C[M]$ stands for the term built up as the context C and where M replaces the hole. Morris's equivalence $\simeq_{\lambda}$ is then induced by the following testing preorder:

$$(3) \qquad M \sqsubseteq_{\lambda} N \text{ iff } \forall \text{ context } C \text{ closing } M \text{ and } N \text{ . } C[M] \Downarrow \Rightarrow C[N] \Downarrow \text{ .}$$

In what regards π-calculus, besides the standard operational semantics given in terms of a labeled transition system, used to define the notion of bisimulation, a reduction semantics can be associated to it following the Chemical Abstract Machine approach [2]; we leave the formal definition for section 3 of the paper (see also [11].) We should only retain here that convergence means the ability to communicate with the environment, after zero or more reduction steps. Hence, one can deal with a contextual preorder of the form

$$(4) \qquad P \sqsubseteq_{\pi} Q \text{ iff } \forall \text{ context } C \text{ . } C[P] \Downarrow \Rightarrow C[Q] \Downarrow \text{ .}$$

In the second half of the paper we give an encoding of λ_j into π, and prove the following adequacy result:

$$(5) \qquad [\![M]\!] \sqsubseteq_{\pi} [\![N]\!] \Rightarrow M \sqsubseteq_{\lambda} N$$

We give for λ_j a presentation involving a semi-explicit manipulation of substitutions; abstractions and variables are now closures of the form $< \lambda x.M, \sigma >$ and $< x, \sigma >$ respectively, where σ is a substitution composed of entries like $[N/y]$ or it is empty (ε). Without entering in the details, the β-contraction is now written as

$$< \lambda x.M, \sigma > N \rightarrow M[N/x]\sigma \text{ ,}$$

and the following new kind of contraction is added for getting the value of variables:

$$< x, [M/x]\sigma > \rightarrow M \text{ .}$$

The structural rule associated to it is

$$\text{If } < x, \sigma > \rightarrow M \text{ then } < x, [N/y]\sigma > \rightarrow M \text{ .}$$

Finally, we specify the meaning of applying a substitution to a term, $M\sigma$, through an equality relation $=_s$ that essentially distributes σ to the subterms of M and that at the variable level makes the following:

$$< x, \rho > \sigma =_s < x, \rho \circ \sigma >$$

$$\epsilon \circ \sigma =_s \sigma$$

$$([N/y]\rho) \circ \sigma =_s [N\sigma/y](\rho \circ \sigma)$$

The encoding of λ_j follows that of [11] for lazy λ. We extend it by representing substitutions as contexts where the terms are placed. The proof of (5) follows from what is called the computational adequacy property, namely

$$M \Downarrow \Leftrightarrow [\![M]\!] \Downarrow .$$

This in turn is shown by combining the following two fundamental properties:

- The substitution process of the lambda calculus is properly mapped by the encoding. If A, M are terms of λ_j such that $A =_s M\sigma$ then the translations of A and $M\sigma$ can do exactly the same actions: $[\![A]\!] \sim [\![M\sigma]\!]$.
- Correctness of $[\![\,]\!]$ w.r.t. the reduction rules of λ_j: A reduction sequence from M in λ_j is mapped, up to strong bisimilarity, to a reduction sequence from $[\![M]\!]$ in π.

Let us come back to the question of non-full-adequacy. When we commented on the embedding of lazy λ by Milner, we mentioned counter-examples to the converse of (2) owning to the shape of the encoding. They serve also to prove that our encoding of λ_j does not have the full-adequacy property. In general, every encoding of the λ-calculus is defined so as to reflect β-contractions properly. The way a translation performs the substitution process is highly dependent on the kind of substitution provided by the process calculus. For example, the encoding of λ into CHOCS [16] just sends the argument $[\![N]\!]$ to the body of the abstraction $[\![M]\!]$. Within the π-calculus the substitution process is much more elementary; channel names can be substituted by channel names and nothing else. This leads to a non-atomic description of the λ-calculus substitution process. Indeed, the solution is to store substitution entries as resources and see a free variable as a process which searches for its actual value and consumes a resource in order to get it. Therefore, to perform a substitution in the encoded calculus, at least as many resources as free occurrences of the variable must be present. Hence, π-contexts with less resources than occurrences of variables allows to do partial substitutions, a feature without counter-part in the λ-calculus, and thus to allow to separate the translations of some indistinguishable pair of λ-terms.

The organization of the paper is the following: In the next section we discuss a number of enriched lazy λ-calculi, with parallelism and convergence testing facilities. In section 2.3 we fix the presentation of λ_j. The π-calculus (syntax, reduction and transition systems) is presented in section 3. In section 4 we define the encoding $[\![\,]\!]$ and we state a few properties to be used later and in section 5 we show that the substitution process of the λ-calculus is properly performed in the encoded calculus. Both the correctness of $[\![\,]\!]$ w.r.t. the set of reduction rules of λ_j and the computational adequacy of $[\![\,]\!]$ are shown in section 6, which ends with the proof of adequacy. Further remarks are put together in section 7.

2 Lambda Calculi and Parallel Lambda Calculi.

In this section we present a variant of the lazy λ_j-calculus for parallel functions defined by Boudol in [4] which includes, besides variables and abstractions treated as closures, an internal choice operator $\oplus$, and a convergence testing combinator $\mathbf{C}$. A large part of the section is devoted to motivate the choice of λ_j. We first fix some terminology.

A lambda calculus consists of a syntax of terms together with a notion of reduction to observables, or values, by means of a predicate $\_ \Downarrow \_$. We read $M \Downarrow V$ as *"M converges to (or has) the value V"*. We use $M \Downarrow$ when V is inessential and $M \Uparrow$ to denote divergence. We call Λ the set of terms, and K that of values, ranged over by A, B, M, N, L and J, K, V, W respectively, unless stated otherwise. Contexts are built up as terms possibly with a hole $[\,]$ as a subterm; they are usually denoted by C, D, E. We use the standard notation $C[M]$ to stand for the term constructed as C and where M replaces the hole. Symbols $\Downarrow$, Λ and K will be decorated depending on the calculus.

Our starting point is Abramsky's pure lazy lambda calculus λ [1]. We keep for it the undecorated symbols Λ, K and $\Downarrow$. Its syntax is given by the grammar:

$$(\Lambda) \qquad M ::= x \mid \lambda x.M \mid (MM)$$

with x a variable taken from a denumerable set Var. The sets $fv(M)$ and $bv(M)$ of free and bound variables of M are defined are usual. Terms and contexts are said to be closed whenever they do not contain free variables.

The set K of values is simply the set of abstractions. The meaning of $\Downarrow$, defined on closed terms, is the following:

$$\frac{}{\lambda x.M \Downarrow \lambda x.M} \qquad \frac{M \Downarrow \lambda x.M' \quad M'[N/x] \Downarrow L}{MN \Downarrow L}$$

In what regards its discriminatory power, this calculus is too poor to be translated into the π-calculus in a fully-adequate way. We have argued in the introduction that an important source of non-full-adequacy of the encoding can be eliminated by extending the calculus with a parallel convergence testing combinator $\mathbf{P}$, the combinator of convergence testing $\mathbf{C}$ being therefore definable: $\mathbf{C} = \lambda x.\mathbf{P}xx$. However, the enriched calculus is not quite appealing in itself, except for the fact that it has a fully-abstract model [1]. It furnishes a limited kind of parallelism, rather unnatural from a programming language point of view.

More expressive languages (also with fully-abstract models associated to them) are λ_j and λ_j^{nv}, obtained by Boudol in [4, 5]. Combinators $\mathbf{P}$ and $\mathbf{C}$ are definable in λ_j and λ_j^{nv}. The construction of those calculi is based on the separation of parallelism from convergence testing. In the rest of the section we study the underlying "control mechanisms" and compare the languages. At the end of the discussion we summarize the definition of λ_j.

2.1 Convergence Testing.

Define an augmented language λ_c with $\Lambda_c = \Lambda \cup \{\mathbf{C}\}$ and $\mathrm{K}_c = \mathrm{K} \cup \{\mathbf{C}\}$. Following [14], the convergence predicate $\Downarrow_c$ is given by adding the following clauses to those of the lazy λ calculus (where $\Downarrow$ is replaced by $\Downarrow_c$):

$$\frac{}{\mathbf{C}\Downarrow_c\mathbf{C}} \qquad \frac{M\Downarrow_c\mathbf{C} \quad N\Downarrow_c}{(MN)\Downarrow_c I}$$

What kind of control mechanism should be added to the lazy λ calculus to allow a "natural" representation of $\mathbf{C}$? The answer comes from the observation that $\mathbf{C}M$ acts much like a strict application (i.e. it asks for the convergence of the argument). In a call-by-value setting, $\mathbf{C}$ would be simply $\lambda x.I$, an abstraction independent of its argument which, after application to a value, behaves like I. However, to keep within the calculus lazy and value applications, we are compelled to associate to each of them a distinguished form of abstraction. To this end, let us introduce a new calculus λ_v with two constructors for abstractions, λ and λ^v. A lazy application has the form of $(\lambda x.M)N$ and evaluates to $M[N/x]$ whatever N is. The shape of a value application is $(\lambda^v x.M)N$; it asks for the convergence of N, say to an observable K, to evaluate then to $M[K/x]$. We define:

$$(\Lambda_v) \qquad M ::= x \mid \lambda x.M \mid \lambda^v x.M \mid (MM)$$

$$(\mathrm{K}_v) \qquad K ::= \lambda x.M \mid \lambda^v x.M$$

We add the following rules to those of the lazy λ calculus (considered with $\Downarrow_v$ at the place of $\Downarrow$):

$$\frac{}{\lambda^v x.M\Downarrow_v\lambda^v x.M} \qquad \frac{M\Downarrow_v\lambda^v x.M' \quad N\Downarrow_v N' \; M'[N'/x]\Downarrow_v L}{MN\Downarrow_v L}$$

Calculi λ_c and λ_v are closely related: each calculus can be translated into the other while preserving the convergence property. We note $(\,)^v_c : \Lambda_c \to \Lambda_v$ and $(\,)^c_v : \Lambda_v \to \Lambda_c$ those translations. As we have already remarked, we take $(\mathbf{C})^v_c = \lambda^v x.I$. For the translation of $\lambda^v x.M$, we set $(\lambda^v x.M)^c_v = \lambda x.(\mathbf{C}x)(M)^c_v$. The rest is identical for both translations; using $(\,)$ to denote them, we define:

$$\begin{array}{lcl} (x) & = & x \\ (\lambda x.M) & = & \lambda x.(M) \\ (MN) & = & (M)\,(N) \end{array}$$

Remark 2.1 *The translation from λ_v to λ_c is related to that from the eager λ calculus to λ_c by Ong [15], which directly codes application as $\overline{MN} = \boldsymbol{C}\overline{N}(\overline{MN})$.*

Proposition 2.2 ***(Mutual simulation of call-by-value and convergence testing)***

1. *For any $M \in \Lambda_c$, $M\Downarrow_c \Leftrightarrow (M)^v_c\Downarrow_v$*
2. *For any $M \in \Lambda_v$, $M\Downarrow_v \Leftrightarrow (M)^c_v\Downarrow_c$*

Proof. The proof of property (1) follows by induction on the length of the derivations $M\Downarrow_c$ and $(M)^v_c\Downarrow_v$ with the help of $(M[N/x])^v_c = (M)^v_c[(N)^v_c/x]$. We sketch here the proof of $(M)^v_c\Downarrow_v \Rightarrow M\Downarrow_c$. To this end, we show more precise implications

(1a) $(M)^v_c \Downarrow_v \lambda x.M' \Rightarrow M\Downarrow_c\lambda x.M''$ and $(M'')^v_c = M'$

(1b) $(M)^v_c \Downarrow_v \lambda^v x.M' \Rightarrow M\Downarrow_c\mathbf{C}$ and $M' = I$

The base case is immediate. Assume $(M)^v_c \Downarrow_v K$ in $k > 0$ steps. Therefore, $(M)^v_c = (M_1M_2)^v_c$ for some M_1 and M_2. Two cases arise, depending on the last rule applied.

(lazy application): $(M_1)^v_c\Downarrow_v\lambda x.M_1'$ and $M_1'[(M_2)^v_c/x]\Downarrow_v K$, both deductions of length less than k.

By i.h. $M_1\Downarrow_c\lambda x.N_1'$ and $(N_1')^v_c = M_1'$. On the other hand, $(N_1'[M_2/x])^v_c = (N_1')^v_c[(M_2)^v_c/x]$ $= M_1'[(M_2)^v_c/x]$. Therefore, if $K = \lambda x.M'$, **(1a)** holds for $N_1'[M_2/x]$ by i.h.. That is, $N_1'[M_2/x]\Downarrow_c\lambda x.M''$ and $(M'')^v_c = M'$. So, **(1a)** holds for $M = (M_1M_2)$ too. If instead $K = \lambda^v x.M'$, case **(1b)** holds for $N_1'[M_2/x]$ and so does for M.

(value application): $(M_1)^v_c\Downarrow_v\lambda^v x.M_1'$, $(M_2)^v_c\Downarrow_v M_2'$ and $M_1'[M_2'/x]\Downarrow_v K$, all deductions of length less than k.

By i.h. $M_1\Downarrow_c\mathbf{C}$, $M_2\Downarrow_c$ and $M_1' = I$. Thence, $M_1'[M_2'/x]$ can only converge to I, so $K = I$. Moreover, $M = M_1M_2\Downarrow_c I$. That means case **(1a)** holds, with $M' = M'' = x$.

We leave the rather technical proof of property (2) for the appendix. It essentially extends that of [15] to our case. □

Even if both translations allow a simulation of convergence in the target calculus, λ_c and λ_v are not completely symmetric. Notice that, from a proof of $(\lambda x.(\mathbf{C}x)M)N\Downarrow_c$, which supposes $N\Downarrow$ and $M[N/x]\Downarrow$ (taking for simplicity $M, N \in \Lambda$), we must be able to construct a proof of $(\lambda^v x.M)N\Downarrow_v$. That is, a proof of $M[N'/x]\Downarrow$ with $N\Downarrow N'$. The question is: why does

$$M[N/x]\Downarrow \text{ and } N\Downarrow N' \Rightarrow (M[N'/x])\Downarrow$$

hold? This is clearly because a term converges (in the sense of $\Downarrow$, $\Downarrow_c$ and $\Downarrow_v$) at most to one value. Therefore, if the convergence of N is used in the proof of $(M[N/x])\Downarrow$, it is N' which is used.

This shows that if λ_c and λ_v are both augmented with the same new operators, defining languages λ_c' and λ_v', the simulation of $\Downarrow_v'$ by $\Downarrow_c'$ is not necessarily preserved since the *at most*

one value for each term property can be lost in λ'_c. In particular, if a non-deterministic operator were added to the calculi, the following would hold:

$$M \Downarrow'_v \not\Leftrightarrow (M)^{c'}_v \Downarrow'_c$$

The fragility of λ_v with regard to the addition of new operators will be confirmed at the end of 2.2 where we show that λ_v cannot incorporate non-determinism safely.

Remark also that to simulate the proof of $\mathbf{C}N\Downarrow_c$, for $N \in \Lambda$, we are lead to prove $(\lambda^v x.I)N\Downarrow_v$ but that not all the premises of the value-application rule are relevant. It does not matter which is the value of N since for all N' we have $I[N'/x]\Downarrow_v$. This suggests the introduction of another calculus, with a strict application instead of a value one. Define λ_s by

$$(\Lambda_s) \qquad M ::= x \mid \lambda x.M \mid \lambda^s x.M \mid (MM)$$

$$(\mathrm{K}_s) \qquad K ::= \lambda x.M \mid \lambda^s x.M$$

$$\frac{}{\lambda x.M \Downarrow_s \lambda x.M} \qquad \frac{}{\lambda^s x.M \Downarrow_s \lambda^s x.M}$$

$$\frac{M\Downarrow_s \lambda x.M' \quad M'[N/x]\Downarrow_s L}{MN\Downarrow_s L} \qquad \frac{M\Downarrow_s \lambda^s x.M' \quad N\Downarrow_s \quad M'[N/x]\Downarrow_s L}{MN\Downarrow_s L}$$

This notion of strict abstraction is exactly what is needed to simulate $\mathbf{C}$ and vice versa in the sense that all the hypothesis to prove the convergence of a term in one calculus are used to prove the convergence of the translated term in the other calculus. Moreover, λ_s accepts the incorporation of non-determinism in an accurate way. Define translations $(\,)^s_c : \Lambda_c \to \Lambda_s$ and $(\,)^c_s : \Lambda_s \to \Lambda_c$ by $(\mathbf{C})^s_c = \lambda^s x.I$ and $(\lambda^s x.M)^c_s = \lambda x.(\mathbf{C}x)(M)^c_s$; the other constructions remain unchanged. A proposition similar to 2.2 holds for them, this time with a straightforward proof:

Proposition 2.3

1. *For any* $M \in \Lambda_c$, $M\Downarrow_c \Leftrightarrow (M)^s_c\Downarrow_s$
2. *For any* $M \in \Lambda_s$, $M\Downarrow_s \Leftrightarrow (M)^c_s\Downarrow_c$

2.2 Parallel Functions.

We review here two ways of introducing parallel facilities in λ following [4, 5], and discuss their integration with convergence testing, strict application and value application.

The first way of gaining a form of parallelism is by means of a parallel composition operator $\parallel$ provided with an interleaving semantics. That is, the evaluation of $(M \parallel N)$ initiates two concurrent sub-computations, one for each component, and this term has a value as soon as one of its components does.

The other form is by adding to the calculus an operator $\oplus$ representing non-determinism. An evaluation of $M \oplus N$ will be either an evaluation of M or an evaluation of N. These extended calculi, apparently quite different, are equivalent from the convergence point of view. Let us first introduce them and then sketch this point.

Call $\lambda_\oplus$ the following lazy λ-calculus extended with $\oplus$:

$$(\Lambda_\oplus) \qquad M ::= x \mid \lambda x.M \mid (MM) \mid (M \oplus N)$$

The set of values, $\mathrm{K}_\oplus$, is simply the set of abstractions. The convergence predicate $\Downarrow_\oplus$ is then defined by adding the following rules to those for the lazy λ calculus (considering that $\Downarrow_\oplus$ is at the place of $\Downarrow$):

$$\frac{M \Downarrow_{\oplus} K}{(M \oplus N) \Downarrow_{\oplus} K} \qquad \frac{N \Downarrow_{\oplus} K}{(M \oplus N) \Downarrow_{\oplus} K}$$

Call λ_p the following lazy λ-calculus extended with $\|$:

$$(\Lambda_p) \qquad M ::= x \mid \lambda x.M \mid (MM) \mid (M \parallel N)$$

$$(\mathrm{K}_p) \qquad K ::= \lambda x.M \mid (K \parallel M) \mid (M \parallel K)$$

The convergence predicate $\Downarrow_p$ is defined by joining the following rules to those for the lazy λ calculus (with $\Downarrow_p$ at the place of $\Downarrow$):

$$\frac{M \Downarrow_p K}{(M \parallel N) \Downarrow_p (K \parallel N)} \qquad \frac{N \Downarrow_p J}{(M \parallel N) \Downarrow_p (M \parallel J)}$$

$$\frac{M \Downarrow_p K \quad N \Downarrow_p J}{(M \parallel N) \Downarrow_p (K \parallel J)} \qquad \frac{M \Downarrow_p (M_0 \parallel M_1) \quad (M_0 N \parallel M_1 N) \Downarrow_p K}{MN \Downarrow_p K}$$

It should be stressed that the introduction of $\|$ or $\oplus$ in the lazy λ calculus changes radically the notion of convergence. Within λ_p a value does not longer coincide with a normal form; if N converges, $(M \parallel N)$ will do even if an infinite computation issues from M. Moreover, since a value can be reached at any stage of the evaluation (and not necessarily at the end), a term can have many values. Similarly, although the values of $\lambda_{\oplus}$ are always normal forms, a term of that language can also have many of them due to the non-deterministic character of $\oplus$; in particular, $(M \oplus N)$ has all values of M and N.

What is the relation between $\|$ and $\oplus$? With the aim of giving some intuitive meaning to $\lambda_{\oplus}$, Boudol says that *"a terminated computation yields only a part of the result. If we were able to join the parts, concurrently evaluated, then we would get the whole result."*. This is precisely what $\|$ enables to do. Therefore, considering

$$(\Lambda_\star) \qquad M ::= x \mid \lambda x.M \mid (MM) \mid (M \star N)$$

to stand both for $\Lambda_{\oplus}$ and Λ_p depending on context and taking two convergence predicates $\Downarrow_p^\star$ and $\Downarrow_{\oplus}^\star$ on $\Lambda_\star$ defined as $\Downarrow_p$ and $\Downarrow_{\oplus}$, replacing $\|$ and $\oplus$ by $\star$ respectively, one can show

$$\text{for any } M \in \Lambda_\star, \quad M \Downarrow_p^\star \Leftrightarrow M \Downarrow_{\oplus}^\star$$

Just remark that $M \Downarrow_p^\star K$ if and only if $K = (\ldots \star \lambda x.N \star \ldots)$. A proof of $M \Downarrow_p^\star K \Leftrightarrow M \Downarrow_{\oplus}^\star \lambda x.N$ follows by a case analysis on the last rule applied for stating the convergence of M in each system.

To achieve the discriminatory power of the lazy λ calculus augmented with $\mathbf{P}$, we must enlarge either λ_p or $\lambda_{\oplus}$ with some "convergence testing facility" in the style of the calculi given in the previous section. The language we have chosen is the result of combining $\lambda_{\oplus}$ and λ_c, called λ_j [4]. The parallel convergence testing combinator is definable in this language: $\mathbf{P} = \lambda x.|lambday.(\mathbf{C}x \oplus \mathbf{C}y)$. It would be equally good to consider λ_s together with $\lambda_{\oplus}$. On the contrary and despite the mutual simulation of λ_c, λ_s and λ_v, the value application is not "compatible" with the non-deterministic choice. That is, the "context lemma" (which states that a functional interpretation is possible) does not hold. Consider the calculus obtained by adding value abstractions to $\lambda_{\oplus}$, denoted $\lambda_{\oplus}^v$. The context lemma for it says that:

$$M \sqsubseteq_A N \;\Rightarrow\; M \sqsubseteq_{\oplus}^v N$$

where $\sqsubseteq_A$ (called applicative preorder) is Morris's preorder but defined upon the applicative tests given by the following grammar:

$$A ::= [] \mid \lambda x.A \mid (AM) \qquad \text{with } M \text{ a closed term}$$

The following example by G. Boudol [7] shows that $\lambda^v_\oplus$ does not satisfy this property. To this end, take the following terms M and N, where $K = \lambda^v x\, \lambda^v y.x$ and $F = \lambda^v x\, \lambda^v y.y$:

$$M = \lambda^v z.(K \oplus F) \qquad N = (\lambda^v z.K) \oplus (\lambda^v z.F)$$

Notice that while M is a value, N is not. One has $M \simeq_A N$ (it is enough to see that $\forall k\, \forall V_1 \ldots V_k\, .\, MV_1 \ldots V_k \Downarrow^v_\oplus \Leftrightarrow NV_1 \ldots V_k \Downarrow^v_\oplus$). Instead, we can built up a non-applicative context C such that $C[M] \Downarrow^v_\oplus$ but $C[N] \Uparrow^v_\oplus$. Define:

$$\begin{cases} \Omega_0 = (\Delta\Delta) & \text{where } \Delta = \lambda x.xx \\ \Omega_{n+1} = \lambda^v x.\Omega_n \end{cases}$$

$$C = (\lambda^v x.((x\Omega_1\Omega_2\Omega_1)(x\Omega_1\Omega_1\Omega_2\Omega_1)))\,[]$$

Remark that $(\Omega_2\Omega_1) \Downarrow^v_\oplus$ and $\Omega_2(\Omega_2\Omega_1) \Downarrow^v_\oplus$ while $(\Omega_1\Omega_1) \Uparrow^v_\oplus$ and $\Omega_1(\Omega_2\Omega_1) \Uparrow^v_\oplus$. Therefore:

$$\begin{array}{rcl} C[M] \Downarrow^v_\oplus & \Leftrightarrow & (M\Omega_1\Omega_2\Omega_1)(M\Omega_1\Omega_1\Omega_2\Omega_1) \Downarrow^v_\oplus \\ & \Leftrightarrow & ((K \oplus F)\Omega_1\Omega_2\Omega_1)((K \oplus F)\Omega_1\Omega_1\Omega_2\Omega_1) \Downarrow^v_\oplus \end{array}$$

To make this hold, we take K for the first instance of $(K \oplus F)$ and F for the second one: after a few reduction steps we get $\Omega_2(\Omega_2\Omega_1) \Downarrow^v_\oplus$. As for $C[N]$, the first step consists in finding a value for N, that is in choosing either its right or its left component to substitute the hole. But neither $C[\lambda^v z.K]$ nor $C[\lambda^v z.F]$ converges: In the first case we get $\Omega_2(\Omega_1\Omega_1) \Uparrow^v_\oplus$ and $\Omega_1(\Omega_2\Omega_1) \Uparrow^v_\oplus$ in the second one. So $C[N] \Uparrow^v_\oplus$.

As pointed out in the previous section, using

$$(C)^c_v = (\lambda x.(\mathbf{C}x)((x\Omega_1\Omega_2\Omega_1)(x\Omega_1\Omega_1\Omega_2\Omega_1)))[]$$

the problem does not arise since x is substituted by the whole term N. The problem does not arise either with $\|$ at the place of $\oplus$ but for a different reason: namely that $(\lambda^v z.K) \,\|\, (\lambda^v z.F)$ is a value. In conclusion, except λ_v and $\lambda_\oplus$, any combination of λ_c, λ_v and λ_s with $\lambda_\oplus$ and λ_p are safe; we could show a context lemma for them.

2.3 The λ_j Calculus.

The λ_j calculus defined in [4] is essentially the union of λ_c and $\lambda_\oplus$. In this work we consider a presentation of λ_j involving "semi-explicit" substitutions; abstractions and variables are given as closures - that is pairs made up of functional values or variables and substitutions, whereas terms as $M\sigma$ with σ a substitution do not belong to the syntax. This choice entails a modification of the notion of evaluation relevant to the adequacy of the encoding.

2.3.1 Syntax.

The syntax of terms and values of λ_j are given by the following grammars:

$$\begin{array}{ll} (\Lambda_j) & M ::= < U, \sigma > \mid (MM) \mid (M \oplus M) \mid \mathbf{C} \\ & U ::= x \mid \lambda x.M \\ (\Sigma_j) & \sigma ::= \varepsilon \mid [M/x]\sigma \\ (\mathrm{K}_j) & K ::= < \lambda x.M, \sigma > \mid \mathbf{C} \end{array}$$

where x stands for a variable taken from a denumerable set Var and σ denotes a substitution, possibly empty (ε). The components $[M/x]$ of a substitution are called *substitution entries*. If σ contains a substitution entry $[M/x]$ then $x \in Var(\sigma)$. The image of a variable by a substitution, $\sigma(x)$, is given by:

$$\varepsilon(x) = x \qquad ([M/y]\sigma)(x) = \begin{cases} M & \text{if } y = x \\ \sigma(x) & \text{otherwise} \end{cases}$$

From now on, closures $< \lambda x.M, \sigma >$ are called abstractions and U is said to be the body of the closure $< U, \sigma >$. We use I to denote the identity $< \lambda x.x, \epsilon >$.

The set $fv(M)$ of *free variables* of a term M is defined as follows:

$$\begin{array}{rcl} fv(< x, \varepsilon >) & = & \{x\} \\ fv(< x, [M/y]\sigma >) & = & fv(< x, \sigma >) \\ fv(< x, [M/x]\sigma >) & = & fv(M) \\ fv(< \lambda x.M, \sigma >) & = & \{y \,/\, \exists z.\, z \in fv(M) - \{x\} \;\&\; y \in fv(\sigma(z))\} \\ fv(\mathbf{C}) & = & \emptyset \\ fv(MN) & = & fv(M) \cup fv(N) \\ fv(M \oplus N) & = & fv(M) \cup fv(N) \end{array}$$

All variables occurring in M that are not free are bound (the set of them is called $bv(M)$). A term M is *closed* if $fv(M) = \emptyset$.

An outer procedure is supposed to exist for dealing with the operations of application of a substitution to a term and composition of substitutions. The meaning of these operations is given by the following clauses:

$$\begin{array}{rcl} (\varepsilon \circ \rho) & =_s & \rho \\ ([N/x]\sigma) \circ \rho & =_s & [N\rho/x](\sigma \circ \rho) \\ < x, \sigma > \rho & =_s & < x, (\sigma \circ \rho) > \\ \mathbf{C}\rho & =_s & \mathbf{C} \\ < \lambda x.M, \sigma > \rho & =_s & < \lambda x.M, (\sigma \circ \rho) > \\ (MN)\rho & =_s & (M\rho \; N\rho) \\ (M \oplus N)\rho & =_s & (M\rho \oplus N\rho) \end{array}$$

We call Λ_{ej} the extension of Λ_j with terms of the form $M\sigma$.

$$\begin{array}{ll} (\Lambda_{ej}) & M ::= < U, \sigma > \mid M\sigma \mid (MM) \mid (M \oplus M) \mid \mathbf{C} \\ & U ::= x \mid \lambda x.M \\ (\Sigma_{ej}) & \sigma ::= \varepsilon \mid [M/x]\sigma \mid (\sigma \circ \sigma) \end{array}$$

The next two statements are easy to prove:

$$\forall \sigma \in \Sigma_{ej}.\, \exists! \zeta \in \Sigma_j.\, \sigma =_s \zeta$$

$$\forall M \in \Lambda_{ej}.\, \exists! N \in \Lambda_j.\, M =_s N$$

We will say that σ denotes ζ, and that M denotes N, respectively.

2.3.2 Evaluation and Reduction.

Two deductive systems are introduced, defining the convergence predicate $\Downarrow_j$ and a (transitive) reduction relation $\rightarrow$ between terms, such that:

$$M \Downarrow K \;\text{ iff } M \rightarrow K$$

$$\mathbf{C}\Downarrow_j\mathbf{C} \qquad < \lambda x.M, \sigma > \Downarrow_j < \lambda x.M, \sigma >$$

$$\frac{M\Downarrow_j\mathbf{C} \quad N\Downarrow_j}{(MN)\Downarrow_j I} \qquad \frac{M\Downarrow_j < \lambda x.M', \sigma > \quad M'[N/x]\sigma\Downarrow_j L}{MN\Downarrow_j L}$$

$$\frac{M\Downarrow_j K}{< x, [M/x]\sigma > \Downarrow_j K} \qquad \frac{< x, \sigma > \Downarrow_j K}{< x, [M/y]\sigma > \Downarrow_j K}$$

$$\frac{M\Downarrow_j K}{(M \oplus N)\Downarrow_j K} \qquad \frac{N\Downarrow_j K}{(M \oplus N)\Downarrow_j K}$$

(β)	$< \lambda x.M, \sigma > N \rightarrow M[N/x]\sigma$
$(Fetch)$	$< x, [M/x]\sigma > \rightarrow M$
(ChL)	$(M \oplus N) \rightarrow M$
(ChR)	$(M \oplus N) \rightarrow N$
(Pop)	If $< x, \sigma > \rightarrow N$ then $< x, [M/y]\sigma > \rightarrow N$
(App)	If $M \rightarrow M'$ then $(MN) \rightarrow (M'N)$
(Obs)	If $N \rightarrow K$ then $\mathbf{C}N \rightarrow I$
$(Trans)$	If $M \rightarrow L$ and $L \rightarrow N$ then $M \rightarrow N$

Remark 2.4 *The relation $\rightarrow$ is not a* one-step *reduction relation (the transitivity rule is part of the system), as it would be if instead of (Obs) and $(Trans)$ we had taken the standard rules for dealing with $\boldsymbol{C}$, namely*

$$\boldsymbol{CC} \rightarrow I$$
$$\boldsymbol{C} < \lambda x.M, \sigma > \rightarrow I$$
$$\textit{If } N \rightarrow N' \textit{ then } (\boldsymbol{C}N) \rightarrow (\boldsymbol{C}N')$$

The rules $(Fetch)$ and (Pop) and the equality $< x, \sigma > \rho =_s < x, (\sigma \circ \rho) >$ describing the behavior of substitutions are new with respect to the definitions of $\rightarrow$ and $=_s$ given in [5]. In both presentations of the calculus substitutions act as ordered environments: the value of a variable x in an environment $\sigma = [M_1/y_1] \ldots [M_n/y_n]\varepsilon$ is taken to be the first M_i, from left to right, for which $x = y_i$. This is defined in [5] at the metalevel by the equation $x\sigma =_s \sigma(x)$.

Finally, let us define the contextual semantics of the language λ_j. A **λ_j-context** is built up as a λ_j-term, possibly with a hole $[]$ in it. The syntax is given by:

$$C ::= [] \mid < x, \sigma > \mid < \lambda x.C, \sigma > \mid (CC) \mid (C \oplus C) \mid \mathbf{C}$$

Remark that the constant $[]$ is not allowed within substitutions.

The operational equivalence of terms $\simeq_\lambda$ is the congruence induced by Morris's testing preorder:

$$M \sqsubseteq_\lambda L \text{ iff } \forall\, C \text{ closing } M \text{ and } L,\ C[M]\Downarrow_j \ \Rightarrow\ C[L]\Downarrow_j$$

Convention 2.5 *Hereafter, we work with closures of Λ_j involving only acceptable substitutions. A substitution $\sigma = [H_1/y_1] \ldots [H_n/y_n]\varepsilon$ is acceptable if $\forall i, j.\ y_j \notin fv(H_i)$.*

Condition of acceptability on substitutions does not affect the expressiveness of closures. For any given arbitrary closure $< V, \sigma >$, a new one $< V', \sigma' >$ is always definable with σ' an acceptable substitution. Substitution σ' is obtained by renaming substitutions entries in σ with fresh variables (i.e. appearing nowhere in the closure); V' is the result of applying that renaming to the free variables of V. The meaning of the original closure is not modified since variables in the body of a closure use at most one entry of the substitution. This restriction, used in section 5, is essential to guarantee that substitutions are correctly performed within the encoded calculus.

3 The π calculus.

The π-calculus of [10] (or more accurately the mini-π calculus of [11]), is defined upon the following syntax of processes:

$$(\Pi) \qquad P ::= \mathbf{0} \mid \overline{x}z.P \mid x(y).P \mid (P|P) \mid !P \mid (\nu y)P$$

where x, y, z are taken from a denumerable set $\mathcal{N}$ of channel names.

The **π-contexts** are defined as follows:

$$C ::= [] \mid \mathbf{0} \mid \overline{x}z.C \mid x(y).C \mid (C|C) \mid !C \mid (\nu y)C$$

We call $\overline{\mathcal{N}}$ the set of overlined channel names and we use n to range over $\mathcal{N} \cup \overline{\mathcal{N}}$. The bar operator on names is idempotent, that is, $\overline{\overline{n}} = n$. We say that x is the *output name* of the process $\overline{x}z.P$ and that $x(y)P$ has x as *input name*. The other constructions, $|$, $!$ and (νy), correspond to parallel composition, replication and restriction on y. Both the input prefix $x(y)$ and the restriction (νy) are binders of y. We do not give the formal definitions of the sets $bn(P)$ and $fn(P)$ of bound and free names of P respectively, which are as usual. The notation $n(P)$ stands for the set of all names occurring in P. In most of the cases we abbreviate $\overline{x}z.\mathbf{0}$ and $x(y).\mathbf{0}$ as $\overline{x}z$ and $x(y)$ respectively. Hereafter, "process" and "term" are used interchangeably.

The reduction system for this calculus is described following the *chemical abstract machine (CHAM) philosophy* [2]. Processes are interpreted as *solutions* (multisets) containing *molecules* (components of the process). The notation for solutions is $\{| m_1, \ldots, m_k |\}$. The machine consists of a few general laws of computation and two kinds of rules between solutions: the irreversible and the structural ones. There is only one irreversible rule $\mapsto$ - also called reaction rule - which accounts for communication, and several structural rules, called $\rightleftharpoons$, concerning the meaning of the operators. For simplicity we omit the most external $\{|\ ,\ |\}$ in the formulation of the rules.

$x(y).P, \overline{x}z.Q$	$\mapsto$	$P[z/y] \mid Q$	$((\nu x)P \mid Q)$	$\rightleftharpoons$	$(\nu x)(P \mid Q)$	if $x \notin fn(Q)$
$P \mid Q$	$\rightleftharpoons$	P, Q	$(\nu x)P$	$\rightleftharpoons$	$(\nu y)P[y/x]$	if $y \notin fn(P)$
$!P$	$\rightleftharpoons$	$P, !P$	$(\nu x)P$	$\rightleftharpoons$	P	if $x \notin fn(P)$
$(\nu x)P$	$\rightleftharpoons$	$(\nu x)\{\mid P \mid\}$	$(\nu x)(\nu y)P$	$\rightleftharpoons$	$(\nu y)(\nu x)P$	
$\mathbf{0}$	$\rightleftharpoons$	$\emptyset$				

The general laws say that we can make computations within sub-solutions of a solution (under restrictions in our case: remark that the sole constructor for which the semantics creates

a sub-solution is (νx)) and also that a computation issued from a solution S_1 can be reproduced on a larger solution $S_1 \uplus S_2$, where $\uplus$ is the multiset union. The process of substitution is left unspecified but it is supposed to accomplish all necessary α-conversions of names.

As an example, let P be the process $x(z)\overline{z}b \mid (\nu a)(\overline{x}a \mid x(y)\overline{y}r)$. Then

$$\begin{array}{rcl} \{\!|\, P \,|\!\} & \stackrel{\star}{\rightleftharpoons} & \{\!|\, x(z)\overline{z}b, (\nu a)\{\!|\, \overline{x}a, x(y)\overline{y}r \,|\!\}\,|\!\} \\ & \mapsto & \{\!|\, x(z)\overline{z}b, (\nu a)\{\!|\, \overline{a}r \,|\!\}\,|\!\} \\ & \stackrel{\star}{\rightleftharpoons} & \{\!|\, x(z)\overline{z}b \mid (\nu a)\overline{a}r \,|\!\} \end{array}$$

And also

$$\begin{array}{rcl} \{\!|\, P \,|\!\} & \stackrel{\star}{\rightleftharpoons} & \{\!|\, (\nu a)\{\!|\, x(z)\overline{z}b, \overline{x}a, x(y)\overline{y}r \,|\!\}\,|\!\} \\ & \mapsto & \{\!|\, (\nu a)\{\!|\, \overline{a}b, x(y)\overline{y}r \,|\!\}\,|\!\} \\ & \stackrel{\star}{\rightleftharpoons} & \{\!|\, (\nu a)\overline{a}b \mid x(y)\overline{y}r \,|\!\} \end{array}$$

As every solution has a process associated with it, $\mapsto$ and $\rightleftharpoons$ will be freely used as relations on processes. We say that the term P *reduces* to Q, $P \triangleright Q$ in notation, whenever $P \stackrel{\star}{\rightleftharpoons} \mapsto \stackrel{\star}{\rightleftharpoons} Q$. In the sequel $\equiv$ will stand for $\stackrel{\star}{\rightleftharpoons}$.

The criterion of convergence is the ability of accepting a communication with the environment, an input or an output, possibly after some reductions. Immediate convergence $\downarrow n$ for $n \in \mathcal{N} \cup \overline{\mathcal{N}}$ is defined as follows:

$$\begin{array}{c} x(y).P \downarrow x \\ \overline{x}z.P \downarrow \overline{x} \end{array} \qquad P \downarrow n \;\Rightarrow\; \left\{ \begin{array}{c} (P|Q) \downarrow n \\ (Q|P) \downarrow n \\ !P \downarrow n \end{array} \right. \qquad P \downarrow n \;\&\; n \neq y \;\Rightarrow\; (\nu y)P \downarrow n$$

We use $P \Downarrow (Q, n)$ as an abbreviated form of $P \stackrel{\star}{\triangleright} Q \;\&\; Q \downarrow n$. The convergence predicate $\Downarrow_\pi$ and the operational preorder $\sqsubseteq_\pi$ are given by:

$$P \Downarrow_\pi \Leftrightarrow_{def} \exists Q\, \exists n \in \overline{\mathcal{N}} \cup \mathcal{N} \,.\, P \Downarrow (Q, n)$$

$$P \sqsubseteq_\pi Q \;\Leftrightarrow_{def}\; \forall\ \pi\text{-context } C \,.\, C[P] \Downarrow_\pi \Rightarrow C[Q] \Downarrow_\pi$$

The congruence induced by $\sqsubseteq_\pi$ is called $\simeq_\pi$. Note that if $P \simeq_\pi Q$ and $Q \Downarrow_\pi$ then $P \Downarrow_\pi$, and also that if $P \triangleright Q$ and $Q \Downarrow_\pi$, then $P \Downarrow_\pi$.

We proceed to define a labeled transition system for the π-calculus following [13], leading to a notion of strong bisimulation which proves to be a congruence. The notation employed is the standard one, we use α to denote the labels of the system, which can be: the bound input $x(y)$, the free or the bound output, $\overline{x}z$ and $\overline{x}(w)$, or the silent transition τ. The sets $n(\alpha)$, $fn(\alpha)$ and $bn(\alpha)$ are defined as for terms.

Let S and S' be two solutions, we say that $S \stackrel{\alpha}{\rightarrow} S'$ holds if and only if one of the following cases applies:

$$\alpha = x(y) \text{ and } \left\{ \begin{array}{l} S \equiv (\nu \vec{u})\{\!|\, x(y)P, \ldots \,|\!\} \\ S' \equiv (\nu \vec{u})\{\!|\, P, \ldots \,|\!\} \end{array} \right. \quad \text{if } x \notin \vec{u} \text{ and } y \text{ occurs only in } P$$

$$\alpha = \overline{x}z \text{ and } \left\{ \begin{array}{l} S \equiv (\nu \vec{u})\{\!|\, \overline{x}z.P, \ldots \,|\!\} \\ S' \equiv (\nu \vec{u})\{\!|\, P, \ldots \,|\!\} \end{array} \right. \quad \text{if } x, z \notin \vec{u}$$

$$\alpha = \overline{x}(z) \text{ and } \left\{ \begin{array}{l} S \equiv (\nu \vec{u})\{\!|\, \overline{x}z.P, \ldots \,|\!\} \\ S' \equiv (\nu \vec{u'})\{\!|\, P, \ldots \,|\!\} \end{array} \right. \quad \text{if } x \notin \vec{u},\ z \in \vec{u} \text{ and } \vec{u'} = \vec{u} - z$$

$\alpha = \tau$ if $S \triangleright S'$

A relation $\mathcal{R}$ between solutions is a *strong bisimulation* if $(S,T) \in \mathcal{R}$ implies that

- whenever $S \stackrel{\alpha}{\rightarrow} S'$ and α is τ, $\overline{x}z$ or $\overline{x}(y)$, for some T', $T \stackrel{\alpha}{\rightarrow} T'$ and $(S',T') \in \mathcal{R}$
- whenever $S \stackrel{x(y)}{\rightarrow} S'$, for some T', $T \stackrel{x(y)}{\rightarrow} T'$ and for all w, $(S'[w/y], T'[w/y]) \in \mathcal{R}$

Two solutions S and T are said strongly bisimilar, written $S \sim T$, if (S,T) is in some strong bisimulation $\mathcal{R}$. Pairs of expressions S and T such that $S \sim T$ are called *bisimulation equivalences*. We will also use the notion of *strong bisimulation up to* $\sim$ which asks only for $(S',T') \in \sim \mathcal{R} \sim$. Strong bisimulations up to $\sim$ are more flexible and are enough for proving strong bisimilarity: If (S,T) is in some strong bisimulation $\mathcal{R}$ up to $\sim$, then $S \sim T$. For a full account of these notions we refer to [9].

Due to the straightforward correspondence between processes and solutions, we will often talk about labeled transitions issued from terms and about bisimulation equivalences made up of terms. The relation between $\sim$ and $\simeq_\pi$ is the following:

Proposition 3.1 *For every pair of processes P, Q,*

1. $\{\!|\, P \,|\!\} \sim \{\!|\, Q \,|\!\} \;\Rightarrow\; P \simeq_\pi Q$
2. $P \simeq_\pi Q \not\Rightarrow \{\!|\, P \,|\!\} \sim \{\!|\, Q \,|\!\}$

Proof.

1. Assume $\{\!|\, P \,|\!\} \sim \{\!|\, Q \,|\!\}$ and $C[P] \Downarrow_\pi$. A straightforward structural induction on C allows to conclude that $C[Q] \Downarrow_\pi$ holds.

   - Base Case ($C = []$): Recall that $P \Downarrow_\pi \Leftrightarrow \exists P', n \,.\, P \stackrel{\star}{\triangleright} P' \downarrow n$ and also that $P \stackrel{\star}{\triangleright} P' \Leftrightarrow \{\!|\, P \,|\!\} \stackrel{\star}{\triangleright} \{\!|\, P' \,|\!\}$. Since $\{\!|\, P \,|\!\} \sim \{\!|\, Q \,|\!\}$, $\exists Q' . \{\!|\, Q \,|\!\} \stackrel{\star}{\triangleright} \{\!|\, Q' \,|\!\}$ & $\{\!|\, P' \,|\!\} \sim \{\!|\, Q' \,|\!\}$. In the case of $n \in \mathcal{N}$, $P' \equiv (\nu \vec{u})(n(y)P_0 | P_1)$ (or simply $P' \equiv (\nu \vec{u})n(y).P_0$) with $n \notin \vec{u}$; hence, $\{\!|\, P' \,|\!\} \stackrel{n(y)}{\rightarrow}$. If $n \in \overline{\mathcal{N}}$ then $P' \equiv (\nu \vec{u})(\overline{n}y.P_0 | P_1)$ (or simply $P' \equiv (\nu \vec{u})\overline{n}y.P_0$). Therefore, either $\{\!|\, P' \,|\!\} \stackrel{\overline{n}(y)}{\rightarrow}$ or $\{\!|\, P' \,|\!\} \stackrel{\overline{n}y}{\rightarrow}$ depending on whether $y \in \vec{u}$ or not.

   -Inductive Case: Follows directly from the fact that $\simeq$ is a congruence and from the inductive hypothesis.

2. The following simple counter-example of the implication makes P and Q differ in the number of silent transitions they can perform:

$$P = (\nu x)(x(y).x(z).u(t).\mathbf{0} \mid (\nu y)(\nu z)\overline{x}y.\overline{x}z.\mathbf{0})$$

$$Q = (\nu x)(x(y).u(t).\mathbf{0} \mid (\nu y)\overline{x}y.\mathbf{0})$$

□

Let us call $SORT(P)$ the ability of process P to communicate with the environment. That is,

$$SORT(P) = \{n \in \mathcal{N} \cup \overline{\mathcal{N}} \;/\; P \downarrow n\}$$

A process P is said *stable* iff one of the following conditions holds:

- $P = \overline{u}z.Q$ or $P = u(x).Q$ or
- $P = (\nu z)Q$ or $P = !Q$ with Q a stable process, or
- $P = Q|R$ with Q and R stable processes and $SORT(Q) \cap \overline{SORT(R)} = \emptyset$

Proposition 3.2 *Let P be a stable process.*

1. $\nexists n . \{n, \overline{n}\} \subseteq SORT(P)$

2. $\nexists P'.\, P \triangleright P'$

Proof.

1. By a straightforward structural induction on P.

2. By structural induction on P. For output, input and restriction processes the proposition is immediate. For the case of parallel composition, the condition on the sorts inhibits communication. For $P = !Q$, with Q stable, assume that some P' exists such that $P \triangleright P'$. That means: Some Q' exists such that

   - either $Q \triangleright Q'$ and $!Q \rightleftharpoons (Q|!Q) \triangleright (Q'|!Q)$

   - or $(Q|Q) \triangleright Q'$, Q' resulting from a communication between the two copies of Q, and $!Q \rightleftharpoons (Q|Q|!Q) \triangleright (Q'|!Q)$

   The first possibility contradicts the i.h. since Q is stable. The second one supposes the existence of some pair $n, \overline{n}$ belonging to $SORT(Q)$, thus contradicting (1) of this proposition.

□

The next proposition establishes sufficient syntactic conditions for the definition of a garbage collection procedure acting on solutions (or terms) to be used in the proofs of preservation of the reduction rules through the encoding. We take it as belonging to the structural rules $\rightleftharpoons$.

Proposition 3.3 *Let P be stable and $SORT(P) = \emptyset$. Then $(P \mid Q) \sim Q$ for every Q.*

Proof. The relation $\mathcal{R}$ defined below is a strong bisimulation:

$$\mathcal{R} = \{(S,T) \,/\, S = (P|Q) \,\&\, T = Q \,\&\, P \text{ is stable } \,\&\, SORT(P) = \emptyset\}$$

Assume $(S,T) \in \mathcal{R}$ and $S \stackrel{\alpha}{\rightarrow} S'$. By definition, $S = (P|Q)$ and $T = Q$. Whatever α is, it can only be performed on Q: Labeled actions $\overline{x}y$, $\overline{x}(y)$ and $x(y)$ are not allowed on P because of the emptiness of $SORT(P)$; silent actions are not allowed either because of proposition 3.2(2) on stable processes. In consequence:

$$\exists Q'.\, Q \stackrel{\alpha}{\rightarrow} Q' \,\&\, S' = (P|Q') \,\&\, T' = Q'$$

We have immediately that $(S', T') \in \mathcal{R}$, thus finishing the proof, for α equal to $\overline{x}y$, to $\overline{x}(y)$ or to τ. For $\alpha = x(y)$, we have to check: $\forall w.\ (S'[w/y], T'[w/y]) \in \mathcal{R}$. This holds due to the side-condition imposed on input transitions; in our case it means that $y \notin n(P)$, so $S'[w/y] = (P|Q'[w/y])$ and $T' = Q'[w/y]$. □

4 The encoding.

The encoding of λ_j into π is a function $[\![-]\!] : (\Lambda_j \times \mathcal{N}) \cup \Sigma \rightarrow \Pi$ following the lines of that given in [11] for the lazy lambda calculus. It is built up with the aim of providing terms of the target calculus suitable to mimic the reduction behavior of terms within the source calculus. To this end, the presence of a channel accompanying the λ-term is crucial. In the λ-calculi, the β-rule specifies the communication of an argument with an abstraction just by juxtaposition; in the π-calculus this communication is achieved through a channel along which $[\![M]\!]$ receives its next argument.

An encoding of λ_j supposes an encoding of substitutions σ. Since substitutions act as environments, we have chosen to encode them as contexts where the terms are placed. In particular, the hole-context [] stands for the empty substitution ε. The notation adopted is the following: $[\![\sigma]\!]$ denotes a context, with one and only one hole in it, and in $[\![\sigma]\!](P)$, the hole is replaced by P.

The translation of closures consists in placing the abstraction or variable in the context of the substitution:

$$[\![< V, \sigma >]\!]u = [\![\sigma]\!]([\![V]\!]u)$$

Clearly, the encoding of substitution entries $[H/x]$ (also called $x := H$) and that of variables x are mutually related; $[H/x]$ will be represented as a resource of the form $x(w)[\![H]\!]w$ while $[\![x]\!]u = \overline{x}u$ outputs to the environment a continuation u. The encoding of $< x, [H/x]\varepsilon >$ consists then in the parallel composition of $x(w)[\![H]\!]w$ and $\overline{x}u$, so it can behave like $[\![H]\!]u$ after a reduction step.

However, the process $(\overline{x}u \mid x(w)[\![H]\!]w \mid x(w)[\![J]\!]w)$ is not an appropriate translation of $[\![< x, [H/x][J/x]\varepsilon >]\!]u$ because it can reduce to $[\![H]\!]u$ as well as to $[\![J]\!]u$. To overcome this problem, input names of substitution entries are made private in a nested fashion which reflects the order of entries, from left to right, in the whole substitution. Thence, $\sigma = [H_1/y_1] \ldots [H_n/y_n]\varepsilon$ is mapped to

$$(\nu y_n)(\,(\nu y_{n-1})(\,\ldots(\nu y_1)(\,[\,] \mid [\![y_1 := H_1]\!]\,) \ldots \mid [\![y_{n-1} := H_{n-1}]\!]\,) \mid [\![y_n := H_n]\!]\,)$$

With this representation, the access to $[\![J]\!]u$ in $(\nu x)(\,(\nu x)(\overline{x}u \mid [\![x := H]\!]) \mid [\![x := J]\!]\,)$ is inhibited because x in $\overline{x}u$ is bound by the nearest (νx). That is, the whole process is α-convertible to $(\nu x)(\,(\nu y)(\overline{y}u \mid [\![y := H]\!]) \mid [\![x := J]\!]\,)$ with y new.

There is yet another point to consider: the non-linear nature of λ-terms. Remark that if M has several free occurrences of x, the resource $[\![x := H]\!]$ must be available for each one of them in the translation of $M[x/H]\varepsilon$. We are led to replicate the substitution entries, that is:

$$[\![x := H]\!] = !x(w)[\![H]\!]w$$

Let us proceed with the encoding of abstractions and applications. Abstractions are mapped to input processes which hide the translated bodies thus preserving the notion of value since input guards are blocking.

The encoding of $\lambda x.M$ is just that proposed by Milner [11]. As he explains, the process $[\![\lambda x.M]\!]u$ receives along u the port name, say z, on which its first argument N will be waiting for a communication. Next it inputs the channel through which $M[N/z]$ can communicate with another potential argument.

$$[\![\lambda x.M]\!]u = u(x).u(v)[\![M]\!]v$$

In a complementary fashion, to simulate the β-rule $(\lambda x.M)N \rightarrow M[N/x]$, the argument sends to the function first a private name z, to substitute x, on which N is available, and then the access name w for the next argument of $[\![(\lambda x.M)N]\!]w$. The substitution $[N/z]$ is not actually performed but rather the body of the abstraction is placed in an environment $(\nu z)([]|[\![z := N]\!])$. Thus every instance of z can consume a copy of the argument (recall that $[\![z := N]\!]$ is a replication of the resource $z(w)[\![N]\!]w$).

$$[\![MN]\!]w = (\nu u)([\![M]\!]u \mid push(N)uw) \quad \text{where}$$

$$push(N)uw = (\nu z)(\overline{u}z.\overline{u}w \mid [\![z := N]\!]) \qquad z \notin fv(N)$$

For the convergence testing combinator we take the definition given in [11]:

$$[\![\mathbf{C}]\!]u = u(x).u(v).(\nu w)\overline{x}w.(\nu y)\overline{w}y.[\![I]\!]v$$

In words, it says that $[\![\mathbf{C}N]\!]u$ behaves first like an ordinary application, (remark that $[\![\mathbf{C}]\!]u$ begins as abstractions do) and then performs successively the following steps:

- it accesses to a copy of N through the private name given for x (the guard $\overline{x}w$ is consumed), thus putting $[\![N]\!]w$ in execution (with w a private name);
- it waits then for the convergence of N, that is, waits for an input process of the form $w(y)\ldots$;

- in such case, it can consume the output guard $\overline{w}y$ leaving the place for $[\![I]\!]v$.

The translation of the non-deterministic choice is very simple; it forces to make a choice between M and N before any other action:

$$[\![M \oplus N]\!]u = (\nu v)(\overline{v}u \mid v(w)[\![M]\!]w \mid v(w)[\![N]\!]w)$$

To sum up, the complete definition of the encoding function is:

$$[\![< V, \sigma >]\!]u = [\![\sigma]\!]([\![V]\!]u)$$

$$[\![\varepsilon]\!] = []$$

$$[\![[M/x]\sigma]\!] = [\![\sigma]\!]((\nu x)([] \mid [\![x := M]\!]))$$

$$[\![x]\!]u = \overline{x}u$$

$$[\![\lambda x.M]\!]u = u(x)u(v)[\![M]\!]u$$

$$[\![MN]\!]w = (\nu u)([\![M]\!]u \mid push(N)uw)$$

$$push(N)uw = (\nu z)(\overline{u}z.\overline{u}w \mid [\![z := N]\!]) \quad z \notin fv(N)$$

$$[\![x := M]\!] = !x(w)[\![M]\!]w$$

$$[\![M \oplus N]\!]u = (\nu v)(\overline{v}u \mid v(w)[\![M]\!]w \mid v(w)[\![N]\!]w)$$

$$[\![\mathbf{C}]\!]u = u(x).u(v).(\nu w)\overline{x}w.(\nu y)\overline{w}y.[\![I]\!]v$$

Two essential features of the encoding, used in the proof of adequacy (sections 5 and 6), are the following:

Proposition 4.1 *For every closed term M,*

1. $M \in \mathrm{K} \Leftrightarrow [\![M]\!]u \downarrow u$.
2. $[\![M]\!]u \stackrel{\star}{\rhd} P \stackrel{\alpha}{\rightarrow}$ *and* $\alpha \neq \tau$ *then* $\alpha = u(v)$.

Proof.

1. By a simple inspection of the definition of $[\![\,]\!]$.
2. By a straightforward induction on M, we can show that for every P, $[\![M]\!]u \stackrel{\star}{\rhd} P$ implies $fn(P) = \{u\} \cup fv(M)$ and $SORT(P) \subseteq \{u\} \cup \overline{fv(M)}$. As a corollary we have that for every closed M and P,

$$[\![M]\!]u \stackrel{\star}{\rhd} P \Rightarrow fn(P) = \{u\} \;\&\; SORT(P) \subseteq \{u\} \tag{1}$$

   To illustrate the kind of reasoning we must do, let us examine the case of closures. Assume $M =< \lambda x.M', \sigma >$. Note that for $\sigma = [H_1/y_1] \ldots [H_n/y_n]$ the acceptability condition on σ, $\forall i, j \,.\, y_j \notin fv(H_i)$, implies $fn([\![\sigma]\!](Q)) = \bigcup_{i=1}^{n} fv(H_i) \cup (fn(Q)/\{y_1, \ldots, y_n\})$. So,

$$fn([\![\sigma]\!]([\![\lambda x.M']\!]u)) = \bigcup_{i=1}^{n} fv(H_i) \cup (fn([\![\lambda x.M']\!]u)/\{y_1, \ldots, y_n\})$$

A simple computation on the definition of $[\![\lambda x.M']\!]u$, and i.h. on M' give us $fn([\![\lambda x.M']\!]u) = \{u\} \cup fv(\lambda x.M')$. Replacing equals by equals in the previous formula for $fn([\![\sigma]\!]([\![\lambda x.M']\!]u))$ we obtain

$$fn([\![\sigma]\!]([\![\lambda x.M']\!]u)) = \bigcup_{i=1}^{n} fv(H_i) \cup \{u\} \cup (fv(\lambda x.M')/\{y_1, \ldots, y_n\})$$

$$\begin{aligned} SORT([\![M]\!]u) &= SORT([\![\lambda x.M']\!]u)/\{y_1, \ldots, y_n, \overline{y_1}, \ldots, \overline{y_n}\} \\ &= (\{u\} \subseteq \{u\} \cup \overline{fv(M)}) \end{aligned}$$

Assume that $[\![M]\!]u \stackrel{\star}{\rhd} P \stackrel{\alpha}{\rightarrow}$ holds. Since by (1) there is no r such that $\overline{r} \in SORT(P)$, α is different from an output transition. Then, it can only be an input transition on some channel w, with $w \in SORT(P)$. As M is closed $w = u$.

□

Finally, note that without the acceptability condition on substitutions, the encoding of substitutions would not be safe. It makes use of replication, a construction that represents quite well persistent features, while substitution in λ-calculus is not actually persistent but performed as many times as needed, after what it vanishes. The encoding of acceptable substitutions reproduces this behavior in the following way:

Proposition 4.2 *Let $[M/x]\sigma$ be an acceptable substitution.*

$$[\![x[M/x]\sigma]\!] \rhd [\![\sigma]\!]((\nu x)([\![M]\!]u \mid [\![x := M]\!])) \equiv [\![M]\!]u$$

Proof. Due to the acceptability hypothesis, x does not belong to the free variables of M so the binding only concerns $[\![x := M]\!]$ and the following holds:

$$[\![\sigma]\!]((\nu x)([\![M]\!]u \mid [\![x := M]\!])) \equiv [\![\sigma]\!]([\![M]\!]u \mid (\nu x)[\![x := M]\!])$$

Applying the law for reduce the scope of a private name repeatedly for all the entries of σ, we have:

$$[\![\sigma]\!]((\nu x)([\![M]\!]u \mid [\![x := M]\!])) \equiv ([\![M]\!]u \mid [\![\sigma]\!]((\nu x)[\![x := M]\!])) \equiv [\![M]\!]u$$

□

5 Substitutions in the encoded calculus.

We focus on the process of substitution, defined in terms of the equalities $=_s$ we gave in section 2.3. As a first step towards the preservation of the reduction rules by $[\![\,]\!]$, we aim at showing that substitutions are properly performed in the encoded calculus. To this end, we extend the definition of $[\![\,]\!]$ to Λ_{ej} thus covering terms of the form $M\sigma$ and composition of substitutions $\sigma \circ \rho$. The new clauses are the following:

$$[\![M\sigma]\!]u = [\![\sigma]\!]([\![M]\!]u)$$

$$[\![\sigma \circ \rho]\!] = [\![\rho]\!]([\![\sigma]\!])$$

Recall that $M\sigma \in \Lambda_{ej}$ denotes the term of Λ_j, say A, where σ has been distributed to the components of M, according to $=_s$. Moreover, $\sigma \in \Sigma_{ej}$ denotes some well-defined substitution $\zeta \in \Sigma_j$. Within the π-calculus, $([\![M\sigma]\!]u, [\![M\zeta]\!]u)$ and $([\![M\sigma]\!]u, [\![A]\!]u)$ are bisimulation equivalences for any term $M \in \Lambda_j$. The proof uses the following properties $(*)$ and $(**)$ and the auxiliary bisimulation equivalences listed below (whose proofs are sketched in the appendix).

$(*)$ If $[H/x]\zeta$ is acceptable then $(N[H/x]\varepsilon)\zeta = N([H/x]\zeta)$

$(**)$ $[\![(M\zeta)\rho]\!]u = [\![M(\zeta \circ \rho)]\!]u$

B1 $(\nu x)(P|Q|!x(w).F) \sim (\nu x)(P|!x(w).F)|(\nu x)(Q|!x(w)F)$ if x occurs in P, Q, F only as output name.

Shown in [12], this bisimilarity enables the distribution of substitution-like processes.

B2 $(\nu x)(v(z).P|!x(w).F) \sim (\nu x)v(z').(P[z'/z]|!x(w).F)$ if $v \neq x$ and z' is a new name.

B3 $(\nu x)(!v(z).P|!x(w).F) \sim (\nu x)!v(z').(P[z'/z]|!x(w).F)$ whenever $v \neq x$ and z' is a new name.

B4 $(\nu x)r(z).P \sim r(z').(\nu x)P[z'/z]$ if $r \neq x$ and z' is a new name.

B5 $(\nu x)!z(w).([\![N]\!]w|[\![x := H]\!]) \sim !(\nu x)z(w).([\![N]\!]w|[\![x := H]\!]$ whenever $z \neq x$.

Theorem 5.1 *Let $M \in \Lambda_{ej}$ and $\sigma \in \Sigma_{ej}$.*

(I) *If $M\sigma =_s A$ with $A \in \Lambda_j$, then $[\![M\sigma]\!]u \sim [\![A]\!]u$.*

(II) *If $\sigma =_s \zeta$ with $\zeta \in \Sigma_j$, then $\forall M \in \Lambda_{ej}.\ [\![M\sigma]\!]u \sim [\![M\zeta]\!]u$.*

Proof.
(I) To prove this proposition, we prove that the relation $\sim_s$ made up of the following pairs is a strong bisimulation up to $\sim$:

$$
\begin{array}{ll}
1: & ([\![(MN)\sigma]\!]u\ ,\ [\![(M\sigma\ N\sigma)]\!]u\ ) \\
2: & ([\![(M \oplus N)\sigma]\!]u\ ,\ [\![(M\sigma \oplus N\sigma)]\!]u\ ) \\
3: & ([\![< \lambda x.M, \eta > \sigma]\!]u\ ,\ [\![< \lambda x.M, \eta \circ \sigma >]\!]u\ ) \\
4: & ([\![\mathbf{C}\sigma]\!]u\ ,\ [\![\mathbf{C}]\!]u\ ) \\
5: & ([\![< x, \eta > \sigma]\!]u\ ,\ [\![< x, \eta \circ \sigma >]\!]u\ )
\end{array}
$$

1: Two cases can arise, depending on whether σ is a composed substitution or not.
(a) Assume $\sigma = [H/x]...\varepsilon$. We show the proposition by induction on σ. If $\sigma = \varepsilon$ the proposition is immediate. Also is $[\![\varepsilon]\!](push(N)wu) \sim push(N\varepsilon)wu$. Consider that $\sigma = [H/x]\zeta$. The inductive hypothesis says that $([\![(MN)\zeta]\!]u\ ,\ [\![(M\zeta\ N\zeta)]\!]u\ )$ and $[\![\zeta]\!](push(N)wu) \sim push(N\zeta)wu$ hold for any pair of terms M, N belonging to Λ_{ej}.

Let us first show $[\![\sigma]\!](push(N)wu) \sim push(N\sigma)wu$. One has, for r and v fresh names:

$$
\begin{array}{rcl}
(\nu x)(\ push(N)wu \,|\, [\![x := H]\!]\ ) & = & (\nu x)(\ (\nu z)(\overline{w}z.\overline{w}u|[\![z := N]\!])\,|\,[\![x := H]\!]\ ) \\
 & \equiv & (\nu z)(\nu x)(\ \overline{w}z.\overline{w}u|[\![z := N]\!]|[\![x := H]\!]\ ) \\
\text{by (B1)} & \sim & (\nu z)(\ (\nu x)(\overline{w}z.\overline{w}u|[\![x := H]\!])\,|\,(\nu x)([\![z := N]\!]|[\![x := H]\!])\ ) \\
 & \equiv & (\nu z)(\ \overline{w}z.\overline{w}u\,|\,(\nu x)([\![z := N]\!]|[\![x := H]\!])\ ) \\
\text{by (B3)} & \sim & (\nu z)(\ \overline{w}z.\overline{w}u\,|\,(\nu x)!z(r).([\![N]\!]r|[\![x := H]\!])\ ) \\
\text{by (B5)} & \sim & (\nu z)(\ \overline{w}z.\overline{w}u\,|\,!(\nu x)z(r).([\![N]\!]r|[\![x := H]\!])\ ) \\
\text{by (B4)} & \sim & (\nu z)(\ \overline{w}z.\overline{w}u\,|\,!z(v).(\nu x)([\![N]\!]v|[\![x := H]\!])\ ) \\
 & = & (\nu z)(\ \overline{w}z.\overline{w}u\,|\,!z(v).[\![N[H/x]\varepsilon]\!]v\ ) \\
 & = & push(N[H/x]\varepsilon)wu
\end{array}
$$

Therefore:

$$
\begin{array}{rcl}
[\![\sigma]\!](push(N)wu) & = & [\![\zeta]\!]((\nu x)(\ push(N)wu \,|\, [\![x := H]\!]\ )) \\
 & \sim & [\![\zeta]\!](push(N[H/x]\varepsilon)wu) \\
\text{by i.h.} & \sim & push((N[H/x]\varepsilon)\zeta)wu) \\
 & = & (\nu z)(\overline{w}z.\overline{w}u|!z(r).[\![(N[H/x]\varepsilon)\zeta]\!]r) \\
\text{by } (*) & = & (\nu z)(\overline{w}z.\overline{w}u|!z(r).[\![\sigma]\!]([\![N]\!]r)) \\
 & = & push(N\sigma)wu
\end{array}
$$

The proof of 1 is then as follows:

$$\begin{array}{rcl}
[\![(MN)\sigma]\!]u & = & [\![\zeta]\!]((\nu x)(\,(\nu w)([\![M]\!]w|push(N)wu)\,|\,[\![x := H]\!])) \\
 & \equiv & [\![\zeta]\!]((\nu w)(\nu x)([\![M]\!]w|push(N)wu|[\![x := H]\!])) \\
\text{by (B1)} & \sim & [\![\zeta]\!]((\nu w)(\,(\nu x)([\![M]\!]w|[\![x := H]\!])\,|\,(\nu x)(push(N)wu|[\![x := H]\!])\,)) \\
 & = & [\![\zeta]\!]((\nu w)([\![M[H/x]\varepsilon]\!]w\,|\,(\nu x)(push(N)wu|[\![x := H]\!])\,)) \\
 & \sim & [\![\zeta]\!]((\nu w)([\![M[H/x]\varepsilon]\!]w\,|\,push(N[H/x]\varepsilon)wu)) \\
 & = & [\![\zeta]\!]([\![M[H/x]\varepsilon\, N[H/x]\varepsilon]\!]u) \\
\text{by i.h.} & \sim & [\![(M[H/x]\varepsilon)\zeta\,(N[H/x]\varepsilon)\zeta]\!]u \\
\text{by } (*) & = & [\![M\sigma\, N\sigma]\!]u
\end{array}$$

(b) $\sigma = \zeta \circ \rho$. The proof is by induction on the number n of occurrences of $\circ$ in σ, and uses the result shown in (a) for substitutions not made up of compositions. One has, in general:

$$[\![(MN)(\zeta \circ \rho)]\!]u = [\![\zeta \circ \rho]\!]([\![MN]\!]u) = ([\![\rho]\!]([\![\zeta]\!])([\![MN]\!]u) = [\![\rho]\!]([\![\zeta]\!]([\![MN]\!]u))$$

If $n = 1$, using (a) twice, once for ζ and once for ρ, and then property $(**)$, we obtain:

$$[\![\rho]\!]([\![\zeta]\!]([\![MN]\!]u)) \sim [\![(M\zeta)\rho\,(N\zeta)\rho]\!]u \sim [\![M\sigma\, N\sigma]\!]u$$

If $n > 1$ owing to ζ non empty (or to ρ non empty but not both), i.h. together with (a) allow to conclude the theorem. If both substitutions are compositions, it holds by a double application of the inductive hypothesis.

For the remaining bisimulation equivalences, we detail only the proof for substitutions of the form $[H/x]\varepsilon$. The extension to $[H/x]\zeta$ and the way of dealing with composed substitutions are similar to those for the application case.

2:

$$\begin{array}{rcl}
[\![(M \oplus N)\sigma]\!]u & = & (\nu x)((\nu v)(\overline{v}u|v(w).[\![M]\!]w|v(w).[\![N]\!]w)|[\![x := H]\!]) \\
 & \equiv & (\nu v)(\nu x)(\overline{v}u|v(w).[\![M]\!]w|v(w).[\![N]\!]w|[\![x := H]\!]) \\
\text{by (B1)} & \sim & (\nu v)(\,(\nu x)(\overline{v}u|[\![x := H]\!])\,| \\
 & & (\nu x)(v(w).[\![M]\!]w|[\![x := H]\!])\,|\,(\nu x)(v(w)[\![N]\!]w|[\![x := H]\!])\,) \\
 & \equiv & (\nu v)(\,\overline{v}u\,|\,(\nu x)(v(w).[\![M]\!]w|[\![x := H]\!])\,|\,(\nu x)(v(w).[\![N]\!]w|[\![x := H]\!])\,) \\
\text{by (B2)} & \sim & (\nu v)(\,\overline{v}u\,|\,(\nu x)v(w).([\![M]\!]w|[\![x := H]\!])\,|\,(\nu x)v(w).([\![N]\!]w|[\![x := H]\!])\,) \\
\text{by (B4)} & \sim & (\nu v)(\,\overline{v}u\,|\,v(w).(\nu x)([\![M]\!]w|[\![x := H]\!])\,|\,v(w).(\nu x)([\![N]\!]w|[\![x := H]\!])\,) \\
 & = & (\nu v)(\,\overline{v}u\,|\,v(w).[\![M\sigma]\!]w\,|\,v(w).[\![N\sigma]\!]\,) \\
 & = & [\![(M\sigma \oplus N\sigma)]\!]u
\end{array}$$

3: The following holds as a consequence of the definition of $[\![\eta \circ \sigma]\!]$:

$$[\![< \lambda x.M, \eta > \sigma]\!]u = [\![\sigma]\!]([\![\eta]\!]([\![\lambda x.M]\!]u)) = ([\![\sigma]\!]([\![\eta]\!]))([\![\lambda x.M]\!]u) = [\![< \lambda x.M, \eta \circ \sigma >]\!]u$$

4: Since $fv(\mathbf{C}) = \emptyset$, one has:

$$[\![\mathbf{C}\sigma]\!]u = (\nu x)(\,[\![\mathbf{C}]\!]u|[\![x := H]\!]\,) \equiv [\![\mathbf{C}]\!]u$$

5: Similar to case 3.

(II) This property follows as a corollary of $[\![M(\varepsilon \circ \rho)]\!]u \equiv [\![M\rho]\!]u$ and $[\![M([H/y]\eta \circ \rho)]\!]u \sim [\![M[H\rho/x](\eta \circ \rho)]\!]u$ whenever $[H/y]\eta$ and ρ are acceptable substitutions. The former equivalence is evident; we show the second one by structural induction on M. Call $\sigma = \varepsilon \circ \rho$ and $\zeta = [H\rho/x](\eta \circ \rho)$.

- Base case: $M =< x, \varepsilon >$. Two cases can happen: If $x \neq y$ then it is easy to see the following:

  $$[\![< x, [H/y]\eta \circ \rho >]\!]u \sim [\![< x, \eta \circ \rho >]\!]u \sim [\![< x, [L/y](\eta \circ \rho) >]\!]u$$

  for any term L such that $fv(L) \cap Var(\eta, \rho) = \emptyset$; by the acceptability hypothesis, this holds in particular for $L = H\rho$.

If $x = y$ we have:

$$[\![< x, [H/x]\eta \circ \rho >]\!]u \sim [\![< x, [H\rho/x]\varepsilon >]\!]u \sim [\![< x, [H\rho/x]\theta >]\!]u$$

for any acceptable θ such that $fv(H\rho) \cap Var(\theta) = \emptyset$. By the acceptability of $[H/y]\eta$ and ρ, this holds for $\theta = \eta \circ \rho$.

- Inductive cases: We sketch the proof for $M =< \lambda x.N, \psi >$. The proof of the other cases is straightforward using the i.h.and **(I)**. We have, using **(I)**:

$$[\![M\sigma]\!]u \sim [\![< \lambda x.N, \psi \circ \sigma >]\!]u = [\![\psi \circ \sigma]\!](u(x).u(v).[\![N]\!]v)$$

Applying bisimulation equivalences B2 and B4, property $(**)$ and i.h., the following holds:

$$[\![M\sigma]\!]u \sim u(x').u(v').[\![(N[x'/x])(\psi \circ \sigma)]\!]v') \sim$$

$$u(x').u(v').[\![((N[x'/x])\psi)[H\rho/x](\eta \circ \rho)]\!]v' \sim u(x').u(v').[\![(N[x'/x])(\psi \circ \zeta)]\!]v' \sim$$

$$[\![\psi \circ \zeta]\!](u(x').u(v').[\![N[x'/x]]\!]v') \equiv [\![\zeta]\!]([\![< \lambda x.N, \psi >]\!]u) = [\![M\zeta]\!]u$$

□

6 Adequacy of the encoding.

The adequacy of the encoding,

$$(Ad) \quad : \quad [\![M]\!]p \sqsubseteq_\pi [\![N]\!]p \Rightarrow M \sqsubseteq_\lambda N,$$

follows essentially from its computational adequacy, that is, $M\Downarrow_j \Leftrightarrow [\![M]\!]u\Downarrow_\pi$. We state first how $[\![\,]\!]$ reflects the complete set of reduction rules of λ_j.

It is worth noticing that some reductions within λ_j do not correspond in the π-calculus to a sequence of $\rhd$ reductions since the substitution indicated by the rule (β) is not actually performed in an encoded term. However, as we proved in the last section, $[\![M\sigma]\!]$ and the term where the substitution is performed can do exactly the same actions (i.e. they are strongly bisimilar). Thus, one can show the following correctness result which entails the *only if* half of the computational adequacy property:

Lemma 6.1 *If $L_1 \rightarrow L_2$ then $[\![L_1]\!]u \rhd^\star \sim [\![L_2]\!]u$ for any channel u.*

Proof. By case analysis on the last rule applied in the deduction of $L_1 \rightarrow L_2$. All the substitutions used here are acceptable.

(β) $L_1 =< \lambda x.M, \sigma > N \rightarrow M[N/x]\sigma =_s L_2$. Assume $(\{x\} \cup Var(\sigma)) \cap fv(N) = \emptyset$.

$$\begin{array}{rcl}
[\![(< \lambda x.M, \sigma >)N]\!]w & = & (\nu u)([\![\sigma]\!]([\![\lambda x.M]\!]u)|push(N)uw) \\
Var(\sigma) \cap fv(N) = \emptyset & \equiv & (\nu u)([\![\sigma]\!]([\![\lambda x.M]\!]u|push(N)uw)) \\
\text{for } z \notin fv(MN) & = & (\nu u)([\![\sigma]\!](u(x).u(v).[\![M]\!]v \mid (\nu z)(\overline{u}z.\overline{u}w \mid [\![z := N]\!])) \\
 & \rhd & (\nu u)([\![\sigma]\!]((\nu z)(u(v).[\![M]\!]v[z/x] \mid \overline{u}w \mid [\![z := N]\!]))) \\
x \notin fv(N) & \equiv & (\nu u)([\![\sigma]\!]((\nu x)(u(v).[\![M]\!]v \mid \overline{u}w \mid [\![x := N]\!]))) \\
 & \rhd & [\![\sigma]\!]((\nu x)([\![M]\!]w \mid [\![x := N]\!]) \\
 & = & [\![M\,[N/x]\sigma]\!]w \\
\text{by theorem 5.1} & \sim & [\![L_2]\!]w
\end{array}$$

$(Fetch)$ $L_1 =< x, [L_2/x]\sigma > \rightarrow L_2$

$$\begin{array}{rcl}
[\![< x, [L_2/x]\sigma >]\!]u & = & [\![\sigma]\!]((\nu x)(\overline{x}u|[\![x := L_2]\!])) \\
 & \rhd & [\![\sigma]\!]((\nu x)([\![L_2]\!]u|[\![x := L_2]\!])) \\
\text{by 4.2, since } [L_2/x]\sigma \text{ is acceptable} & \equiv & [\![L_2]\!]u
\end{array}$$

(ChL) $L_1 = (L_2 \oplus N) \rightarrow L_2$

$$\begin{array}{rcl} [\![(L_2 \oplus N)]\!]u & = & (\nu v)(\overline{v}u \mid v(w).[\![L_2]\!]w \mid v(w).[\![N]\!]w) \\ & \rhd & (\nu v)([\![L_2]\!]u \mid v(w).[\![N]\!]w) \\ & \equiv & [\![L_2]\!]u \mid (\nu v)v(w).[\![N]\!]w \\ \text{by proposition 3.3} & \sim & [\![L_2]\!]u \end{array}$$

(ChR) As before.

(Pop) $L_1 =< x, [M/y]\sigma > \rightarrow L_2$ with $< x, \sigma > \rightarrow L_2$

$$\begin{array}{rcl} [\![< x, [N/y]\sigma >]\!]u & = & [\![\sigma]\!]((\nu y)([\![x]\!]u | [\![y := N]\!])) \\ x \neq y \text{ and proposition } 3.3 & \sim & [\![\sigma]\!]([\![x]\!]u) \\ \text{by i.h.} & \rhd & [\![L_2]\!]u \end{array}$$

(App) $L_1 = (MN) \rightarrow (M'N) = L_2$ with $M \rightarrow M'$

$$\begin{array}{rcl} [\![MN]\!]u & = & (\nu w)([\![M]\!]w \mid (\nu z)(\overline{w}z.\overline{w}u \mid [\![z := N]\!]) \\ \text{by i.h.} & \stackrel{\star}{\rhd}\sim & (\nu w)([\![M']\!]w \mid (\nu z)(\overline{w}z.\overline{w}u \mid [\![z := N]\!]) = [\![M'N]\!]u \end{array}$$

(Obs) $L_1 = \mathbf{C}N \rightarrow I = L_2$ with $N \rightarrow V$

$$\begin{array}{rcl} [\![\mathbf{C}N]\!]u & = & (\nu w)(w(x).w(v).(\nu r)\overline{x}r.(\nu y)\overline{r}y.[\![I]\!]v \mid (\nu z)(\overline{w}z.\overline{w}u \mid [\![z := N]\!])) \\ & \stackrel{\star}{\rhd} & (\nu z)((\nu r)(\overline{z}r.(\nu y)\overline{r}y.[\![I]\!]u) \mid [\![z := N]\!]) \\ & \rhd & (\nu z)(\nu r)([\![N]\!]r \mid (\nu y)\overline{r}y.[\![I]\!]u \mid [\![z := N]\!]) \\ \text{by i.h.} & \stackrel{\star}{\rhd}\sim & (\nu z)(\nu r)([\![V]\!]r \mid (\nu y)\overline{r}y.[\![I]\!]u \mid [\![z := N]\!]) \\ & \rhd & [\![I]\!]u \mid (\nu z)[\![z := N]\!] \mid (\nu r)(\nu y)... \\ \text{by proposition } 3.3 & \sim & [\![I]\!]u \end{array}$$

$(Trans)$ $L_1 \rightarrow L_2$ with $L_1 \rightarrow L$ and $L \rightarrow L_2$

By transitivity of $\rhd$ and $\sim$.

□

Theorem 6.2 ***(Computational Adequacy)***
For every closed $M \in \Lambda_j$, $M\Downarrow_j \Leftrightarrow [\![M]\!]u\Downarrow_\pi$.

Proof.
Case $\Rightarrow$: Suppose that $M\Downarrow_j K$ for some value K, that is, $M \rightarrow K$. One has $[\![M]\!]u \stackrel{\star}{\rhd}\sim [\![K]\!]u$ as a direct application of lemma 6.1. It is easy to see that steps of $\rhd$ can always be moved to the beginning of the reduction, thus giving $[\![M]\!]u \stackrel{\star}{\rhd}\sim [\![K]\!]u$. As we have remarked in 4.1, $[\![K]\!]u \downarrow u$, so $[\![M]\!]u \Downarrow_\pi$.
Case $\Leftarrow$: Based on the remark 4.1, we show the following refined version of the statement:

$$\forall \text{ closed } M\,.\,\forall P\,.\,[\![M]\!]u \stackrel{\star}{\rhd} P \stackrel{u(v)}{\rightarrow} \text{ implies}$$

1. either $P \sim [\![< \lambda x.A, \rho >]\!]u$ and $M \rightarrow < \lambda x.A, \rho >$ or
2. $P \sim [\![\mathbf{C}]\!]u$ and $M \rightarrow \mathbf{C}$.

In the sequel, $\stackrel{n}{\rhd}$ will stand for a reduction of length n. Assume that $[\![M]\!]u \stackrel{n}{\rhd} P \stackrel{u(v)}{\rightarrow}$. The proof follows by induction on n.

Case $n = 0$. Straightforward, as we have remarked in 4.1(1).

Case $n = k + 1$. By case analysis on the structure of M:

- $M =< x, \sigma >$. Since the reduction starting from $[\![M]\!]u$ is non-empty, the reduction from $[\![M]\!]$ has the shape

$$[\![M]\!]u \rhd [\![M']\!]u \stackrel{k}{\rhd} P$$

with M' such that $[M'/x]$ is the first, in a left to right order, substitution entry for x in σ. Applying i.h. to the reduction from $[\![M']\!]u$, one gets

1. either $P \sim [\![< \lambda x.A, \rho >]\!]u$ and $M' \rightarrow < \lambda x.A, \rho >$, or
2. $P \sim [\![\mathbf{C}]\!]u$ and $M' \rightarrow \mathbf{C}$.

Since $M =< x, \sigma > \rightarrow M'$ holds using (Pop) and $(Fetch)$, the result can be extended to M.

- $M = M_1 M_2$. Consider the expanded form $(\nu r)([\![M_1]\!]r \mid push(M_2)ru)$ of $[\![M]\!]u$. In order to bring u to the surface, $push(M_2)ru$ must be consumed; so, the reduction from $[\![M]\!]u$ can be split as follows:

$$\begin{array}{lll} (\nu r)([\![M_1]\!]r \mid push(M_2)ru) & \stackrel{n_1}{\rhd} & (\nu r)(R \mid push(M_2)ru) = S \\ & \stackrel{n_2}{\rhd} & P \end{array}$$

where $n = n_1 + n_2$, $n_1 \geq 0$ and $[\![M_1]\!]r \stackrel{n_1}{\rhd} R \stackrel{r(v)}{\rightarrow}$. By i.h. one has

- either $S \sim [\![< \lambda x.B, \delta > M_2]\!]u$ with $M_1 \rightarrow < \lambda x.B, \rho >$, or
- $S \sim [\![\mathbf{C}M_2]\!]u$ with $M_1 \rightarrow \mathbf{C}$.

If the first case holds, there exists $P' \stackrel{u(v)}{\rightarrow}$ such that

$$S \sim [\![< \lambda x.B, \delta > M_2]\!]u \stackrel{n_2}{\rhd} P' \sim P$$

Furthermore, the reduction from $[\![< \lambda x B, \delta > M_2]\!]u$ can only begin with a simulation of the β-rule. Let $M' \in \Lambda_j$ be such that $M' =_s B[M_2/x]\delta$; lemma 6.1 allows to say

$$[\![< \lambda x.B, \delta > M_2]\!]u \stackrel{+}{\rhd} \sim [\![M']\!]u \stackrel{n_3}{\rhd} P'$$

Note that $n_3 < n$. So i.h. on the reduction from $[\![M']\!]u$ yields

1. either $P \sim P' \sim [\![< \lambda x.A, \rho >]\!]u$ and $M \rightarrow (< \lambda x.B, \delta >)M_2 \rightarrow M' \rightarrow < \lambda x.A, \rho >$, or
2. $P \sim P' \sim [\![\mathbf{C}]\!]u$ and $M \rightarrow (< \lambda x.B, \delta >)M_2 \rightarrow M' \rightarrow \mathbf{C}$ holds.

If instead we are in the second case, for some $P' \stackrel{u(v)}{\rightarrow}$ one has

$$S \sim [\![\mathbf{C}M_2]\!]u \stackrel{n_2}{\rhd} P' \sim P$$

Moreover, the reduction from $[\![\mathbf{C}M_2]\!]u$ must be of the following shape:

$$[\![\mathbf{C}M_2]\!]u \stackrel{n_4}{\rhd} (\nu z)(\nu r)(T \mid (\nu y)\overline{r}y.[\![I]\!]u \mid [\![z := M_2]\!]) \stackrel{+}{\rhd} P' \stackrel{u(v)}{\rightarrow}$$

To allow an input transition on u, $[\![M_2]\!]r \stackrel{+}{\rhd} T \stackrel{r(v)}{\rightarrow}$. So inductive hypothesis says that $P \sim P' \sim [\![I]\!]u$. Finally, $M \rightarrow (\mathbf{C}M_2) \rightarrow I$ since by i.h.

- either $T \sim [\![< \lambda x.L, \psi >]\!]r$ and $M_2 \rightarrow < \lambda x.L, \psi >$, or
- $T \sim [\![\mathbf{C}]\!]r$ and $M_2 \rightarrow \mathbf{C}$ holds.

- $M = (M_1 \oplus M_2)$. The encoding of M forces any reduction from $[\![M]\!]u$ to chose first the branch it wants to pursue, that is,

$$[\![M]\!]u \rhd [\![M_i]\!]u \stackrel{+}{\rhd} P$$

for $i = 1$ or $i = 2$. The theorem holds by a direct application of i.h. to M_i and the fact that $M \rightarrow M_i$ using (ChR) or (ChL).

□

Proposition 6.3 *For every pair of λ-terms M and N, if $[\![M]\!]p \sqsubseteq_\pi [\![N]\!]p$ then $[\![C[M]]\!]p \sqsubseteq_\pi [\![C[N]]\!]p$ for every λ-context C.*

Proof. The compositional definition of the translation guarantees that for any λ-context C, there is a π-context D and a channel name u such that $[\![C[M]]\!]p = D[[\![M]\!]u]$. Assume $E[[\![C[M]]\!]p]\Downarrow_\pi$, for E a π-context. Therefore $E[D[[\![M]\!]u]]\Downarrow_\pi$. From the hypothesis we have $E[D[[\![N]\!]u]]\Downarrow_\pi$, so $E[[\![C[N]]\!]p]\Downarrow_\pi$.

□

Finally, one has (Ad) as a direct combination of 6.2 and 6.3:

Theorem 6.4 ***(Adequacy)***
For every pair of λ-terms M and N, if $[\![M]\!]p \sqsubseteq_\pi [\![N]\!]p$ then $M \sqsubseteq_\lambda N$.

Proof. Consider C such that $C[M]\Downarrow_j$. By theorem 6.2 one has $[\![C[M]]\!]p\Downarrow_\pi$, and by proposition 6.3, $[\![C[M]]\!]p \sqsubseteq_\pi [\![C[N]]\!]p$. Hence, $[\![C[N]]\!]p\Downarrow_\pi$ and, using theorem 6.2 again, $C[N]\Downarrow_j$ holds.

□

7 Further Remarks.

Let us clarify some points remarked in the previous sections. Concerning the converse of (Ad), we mentioned in the introduction a counter-example which exploits the fact that partial substitutions can be described in the encoded calculus. One of the π-contexts C that separates $[\![x(\lambda y.xy)]\!]u$ from $[\![xx]\!]u$, while $x(\lambda y.xy) \sqsubseteq_\lambda xx$, is

$$C = (\nu x)([\,]\,|\,x(w).[\![I]\!]w)$$

since we have

$$C[[\![M]\!]u] \rhd (\nu x)([\![I]\!]w\,|\,push(x)wu) \stackrel{\star}{\rhd} (\nu x)[\![x]\!]u \Uparrow_\pi$$

$$C[[\![N]\!]u] \rhd (\nu x)([\![I]\!]w\,|\,push(\lambda y.xy)wu) \stackrel{\star}{\rhd} (\nu x)[\![\lambda y.xy]\!]u \downarrow u$$

A natural question is how does $\lambda x.x(\lambda y.xy) \not\simeq^{\uplus} \lambda x.xx$? This is simply because taking $L = \uplus I$ in the first clause of the definition of $\simeq^{\uplus}$ one should be able to show $\lambda y.(\uplus I)y \simeq^{\uplus} \uplus I$; since $\uplus I$ reduces to Ω - which will never became an abstraction, this does not hold.

In section 2 we remarked that the rules of $\rightarrow$ for dealing with $\mathbf{C}$ were not the usual ones. It should be clear that, keeping $[\![\,]\!]$ unchanged, the rule

$$\text{If } N \rightarrow N' \text{ then } (\mathbf{C}N) \rightarrow (\mathbf{C}N')$$

does not correspond, in the encoded version, to a sequence of $\rhd$ reductions, thus breaking the correctness result 6.1. To allow an evaluation of $[\![N]\!]w$, $[\![\mathbf{C}N]\!]u$ must consume some guards of $\mathbf{C}$. Therefore it cannot end in a term of the form $[\![\mathbf{C}N']\!]u$.

Also calculi involving $\|$ were put aside owing to the properties of the encodings they induce. At first sight, $(M \parallel N)$ can be encoded as $([\![M]\!]u \,|\, [\![N]\!]u)$, but this results problematic in

the following two aspects. Observe that taken this encoding, the rule of distribution of the argument $(M \parallel N)L \rightarrow (ML \parallel NL)$ does not have associated, within the π-calculus, a sequence of reductions, but rather the strong bisimulation equivalence $(M \parallel N)L \sim (ML \parallel NL)$. Certainly, one can modify the reduction system to fit the encoding by adding rules of the style $(< \lambda x.M, \sigma > \parallel N)L \rightarrow M[L/x]\sigma \parallel (NL)$. A more relevant property of this encoding is the following: $[\![\lambda x.(M \parallel N)]\!]u$ can be distinguished from $[\![\lambda x.M \parallel \lambda x.N]\!]u$ while $\lambda x.(M \parallel N) \simeq_\lambda \lambda x.M \parallel \lambda x.N$. That is, the π calculus allows to count the number of parallel compositions in a term. Hence, additional counter-examples to the converse of adequacy arise.

Another point to stress is that the notion of reduction $\triangleright$ of the π-calculus seems to be more generous than needed. We mean by this that restricting communication to act on private names would not affect our proofs. The new formulation of $\mapsto$ to consider would be

$$(\nu \vec{v})\{|\ ..., x(y).R', \overline{x}z.T, ...\ |\} \rightharpoonup (\nu \vec{v})\{|\ ..., R'[z/y], T, ...\ |\} \qquad \text{with } x \in \vec{v}$$

Acknowledgments.

I am greatly indebted to Gérard Boudol for introducing me to the issues addressed in this paper; this work could have not been done without his suggestions and help. I would like to thank him and Pierre-Louis Curien for carefully reading several drafts of this paper and for their useful comments.

References

[1] S. Abramsky . **The Lazy Lambda Calculus**. In Research Topics in Functional Programming, ed. D. Turner, Addison Wesley. 1989.

[2] G. Berry, G. Boudol. **The Chemical Abstract Machine**. TCS 96(1). 1992.

[3] G. Boudol . **Towards a lambda-calculus for concurrent and communicating systems**. In TAPSOFT 1989, LNCS 351. 1989

[4] G. Boudol. **A Lambda-Calculus for Parallel Functions**. Rapport de Recherche INRIA 1231. 1990.

[5] G. Boudol. **Lambda-Calculi for (strict) Parallel Functions**. Rapport de Recherche INRIA 1387. 1991. To appear in Information and Computation.

[6] G. Boudol. **Asynchrony and the π-calculus**. Rapport de Recherche INRIA 1702. 1992.

[7] G. Boudol. Private communication. 1993.

[8] K. Honda, M. Tokoro. **On Asynchronous Communication Semantics**. LNCS 612. 1992.

[9] R. Milner. **Communication and Concurrency**. Prentice Hall. 1989.

[10] R. Milner, J. Parrow, D. Walker. **A Calculus of Mobile Processes, Parts I and II**. Information and Computation, 100. 1992.

[11] R. Milner. **Functions as Processes**. Math. Struct. in Computer Science, 2. 1992.

[12] R. Milner. **The polyadic π-calculus: a tutorial**. Technical Report ECS-LFCS 91-180, Edimbourg University. 1991.

[13] R. Milner, J. Parrow, D. Walker. **Modal Logics for Mobile Processes**. Technical Report ECS-LFCS 91-136, Edimbourg University. 1991.

[14] C.-H. Luke Ong. **Fully Abstract Models of the Lazy Lambda Calculus**. In Proceedings of the 29th Conference on Foundations of Computer Science. The Computer Science Press. 1988.

[15] C.-H. Luke Ong. **The Lazy Lambda Calculus/ An Investigation into the Foundations of Functional Programming**. PhD Thesis, Imperial College. 1988.

[16] B. Thomsen. **A Calculus of Higher-order Communicating Systems**. Proceedings of the 16th Annual Symposium on Principles of Programming Languages. 1989.

[17] D. Sangiorgi. **Expressing Mobility in Process Algebras: First Order and Higher Order Paradigms**. PhD. Thesis, Department of Computer Science, Edinburgh University. 1993.

[18] D. Sangiorgi. **The Lazy Lambda Calculus in a Concurrency Scenario**. Proceedings of LICS 1992.

Appendix A

In this appendix we tackle the proof of 2.2(2), which says that convergence in the language λ_v can be simulated in λ_c. Let us recall the translation

$$\begin{array}{lcl} (x)^c_v & = & x \\ (\lambda x.M)^c_v & = & \lambda x.(M)^c_v \\ (\lambda^v x.M)^c_v & = & \lambda x.(\mathbf{C}x)(M)^c_v \\ (MN)^c_v & = & (M)^c_v\,(N)^c_v \end{array}$$

The statement to show is:

$$\text{For any } M \in \Lambda_v\ ,\ M{\Downarrow_v} \ \Leftrightarrow\ (M)^c_v{\Downarrow_c}$$

Our proof is adapted from that of [15] given for the simulation of the eager lambda calculus into λ_c. We first introduce some few auxiliary notions and state various results; for proofs not given here we refer to [15].

Define two reduction relations, $\rightarrow_v$ on Λ_v and $\rightarrow_c$ on Λ_c, as follows:

(β)	$(\lambda x.M)N{\rightarrow_v}M[N/x]$
(β^v)	If $K \in \mathrm{K}_v$ then $(\lambda^v x.M)K{\rightarrow_v}M[K/x]$
(Arg)	If $N{\rightarrow_v}N'$ then $(\lambda^v x.M)N{\rightarrow_v}(\lambda^v x.M)N'$
(App)	If $M{\rightarrow_v}M'$ then $(MN){\rightarrow_v}(M'N)$
(β)	$(\lambda x.M)N{\rightarrow_c}M[N/x]$
$(C1)$	$\mathbf{CC}{\rightarrow_c}I$
$(C2)$	$\mathbf{C}(\lambda x.M){\rightarrow_c}I$
$(C3)$	If $M{\rightarrow_c}M'$ then $\mathbf{C}M{\rightarrow_c}\mathbf{C}M'$
(App)	If $M{\rightarrow_c}M'$ then $(MN){\rightarrow_c}(M'N)$

Straightforward inductive arguments allow to show:

$$\text{For } M \in Lv,\ M{\Downarrow_v}K \ \Leftrightarrow M{\stackrel{\star}{\rightarrow}_v}K$$

$$\text{For } M \in Lc,\ M{\Downarrow_c}K \ \Leftrightarrow M{\stackrel{\star}{\rightarrow}_c}K$$

The third reduction relation we need, $\rightarrow_{\beta c} : \Lambda_c \rightarrow \Lambda_c$, enables the use of (β), $(C1)$ and $(C2)$ in any λ_c-context, that is in function position, in argument position and under abstraction. For any $\rightarrow_{\beta c}$-reduction sequence there is a standard reduction sequence composed of a sequence of $\rightarrow_c$-reductions followed by a sequence of $\rightarrow_{\beta c}$-reductions contracting redexes in a left to right fashion.

Definition 7.1 *Let us call* ***lazy contexts*** *those defined by the following grammar:*

$$Q ::= (\mathbf{C}Q)N|(QN)|[]$$

Definition 7.2 *(see 4.4.3.13 in [15])*
Let $M \in \Lambda_c$. An infinite $\rightarrow_{\beta c}$-reduction

$$M = M_0 \rightarrow_{\beta c} M_1 \rightarrow_{\beta c} ... \rightarrow_{\beta c} M_i \rightarrow_{\beta c} ...$$

is ***quasi-lazy*** *if there is an infinite collection of indexes $< n_1, ..., n_i, ... >$ such that for every i, $n_i < n_{i+1}$ and $M_{n_i - 1} \rightarrow_{\beta c} M_{n_i}$ is a $\rightarrow_c$-reduction.*

Remark 7.3 *Let $M \in \Lambda_c$. If M has an infinite quasi-lazy reduction then, for any lazy context Q, $Q[M]$ has an infinite quasi-lazy reduction.*

Proposition 7.4 *(see 4.4.3.18 in [15])*
Let $M \in \Lambda_c$. M has an infinite quasi-lazy reduction if and only if $M \Uparrow_c$.

The next step consists in the introduction of a new reduction relation on Λ_c, $\rightarrow_o$, which simulates $\rightarrow_v$ step-wise. The corresponding rules are just the translation of those defining $\rightarrow_v$:

(1)	If $M \neq (\mathbf{C}x)M'$ then $(\lambda x.M)N \rightarrow_o M[N/x]$
(2)	If $N \in \mathrm{K}_c$ then $(\lambda x.(\mathbf{C}x)M)N \rightarrow_o M[N/x]$
(3)	If $N \rightarrow_o N'$ then $(\lambda x.(\mathbf{C}x)M)N \rightarrow_o (\lambda x.(\mathbf{C}x)M)N'$
(4)	If $M \rightarrow_o M'$ then $(MN) \rightarrow_o (M'N)$

Proposition 7.5 *For $M, N \in \Lambda_c$, $M \rightarrow_o N \Rightarrow M \stackrel{\star}{\rightarrow}_{\beta c} N$.*

Proof. The statement is shown by induction on the length l of the derivation $M \rightarrow_o N$. When $l = 1$, if the rule applied was (1) then $M \rightarrow_{\beta c} N$ just using the (β)-rule. If it was (2), then

$$M = (\lambda x.(\mathbf{C}x)M)N \rightarrow_{\beta c} (\mathbf{C}N)(M[N/x]) \rightarrow_{\beta c} I(M[N/x]) \rightarrow_{\beta c} M[N/x] = N$$

When $l > 1$, the statement follows by a direct combination of i.h. and the fact that $\rightarrow_{\beta c}$-reductions can be performed anywhere in a term. □

By structural induction on the shape of M one can show

$$(M)_v^c[(N)_v^c/x] = (M[N/x])_v^c$$

A straightforward induction on the last rule applied together with the previous property allows to prove the simulation of $\rightarrow_v$ by $\rightarrow_o$:

Proposition 7.6 *For $M, N \in \Lambda_v$, $M \rightarrow_v N \Leftrightarrow (M)_v^c \rightarrow_o (N)_v^c$.*

The determinacy of $\rightarrow_o$ follows as a corollary. Let us introduce some notation:

Notation 7.7 *We use Λ_o for denoting the terms of Λ_c which have the shape of translated Λ_v-terms. That is, $L \in \Lambda_o$ iff $(M)_v^c = L \in \Lambda_c$ for some $M \in \Lambda_v$.*

For $M \in \Lambda_o$, M means that M has a finite $\rightarrow_o$-reduction ending in an irreducible term. Otherwise, $M\uparrow_o$. To name reduction steps we use a decoration like this one: $M \stackrel{\Delta}{\rightarrow}_o M'$. If the rule used is relevant, we write its name above the arrow.

Note that if MN then N is an abstraction; the other value of λ_c, $\mathbf{C}$, is only used applied to arguments. The next lemma is fundamental in the proof of 2.2(2).

Lemma 7.8 *Let $M \in \Lambda_o$. If $M\uparrow_o$ then M has an infinite quasi-lazy reduction.*

Proof. Assume $M \in \Lambda_o$ and $M{\uparrow_o}$, that is

$$M = M_0 \xrightarrow{\Delta_1}_o M_1 \xrightarrow{\Delta_2}_o M_2 \ldots \xrightarrow{\Delta_n}_o M_n \xrightarrow{\Delta_{n+1}}_o \ldots$$

One of the following cases hold:

A. An unbounded number of reductions Δ_i use rule (1) or (2).

B. $\exists N \geq 1 \;\; \forall i \geq N \;\; \Delta_i$ uses (3) as the last rule.

C. $\exists N \geq 1 \;\; \forall i \geq N \;\; \Delta_i$ uses (4) as the last rule.

Suppose case **A** does not hold and that N is the least n such that $\forall i \geq n$. Δ_i uses (3) or (4) as the last rule.

- If Δ_N uses (3) then $M_N = (\lambda x.(\mathbf{C}x)M')L$ where L is not an abstraction; for otherwise Δ_{N+1} would use rule (2), contradicting the assumption. Therefore, (3) is the only candidate for Δ_{N+1}, with premise $L{\rightarrow_o}L'$. For the same reason as before L' cannot be an abstraction. Applying this argument inductively yields case **B** (L does not converge).
- Suppose Δ_N uses (4). Then $M_N = (M_1 M_2)$. If $M_1{\uparrow_o}$ it should be clear that case **C** holds. If not, then $M_1 \lambda x.M_1'$ with $M_1' = (\mathbf{C}x)M_1''$. The case $M_1' \neq (\mathbf{C}x)M_1''$ is inhibited for otherwise rule (1) would be applicable. Moreover, M_2 is not an abstraction so case **B** holds.

If case **A** holds, then M has an infinite quasi-lazy reduction using proposition 7.4. We will consider the other cases. To this end, define the contexts D_{xL} and E_L as follows:

$$D_{xL} = (\lambda x.(\mathbf{C}x)L)[] \qquad E_L = ([]L)$$

and let F, G, H range over D_{xL} and E_L. Cases **B** and**C** can be put together in the following way:

$$M \xrightarrow{\star}_o F[R_0] \rightarrow_o F[R_1] \rightarrow_o \ldots \rightarrow_o F[R_m] \rightarrow_o \ldots$$

Every deduction $F[R_i] \rightarrow_o F[R_{i+1}]$ consists in finitely many uses of rules (3) and (4) and one use of rule (1) or (2). That is, some $n_i \geq 0$, $G_1^i, \ldots G_{n_i}^i$, S_i, S_i' exist such that $S_i \rightarrow_o S_i'$ using either rule (1) or rule (2) and

$$R_i = G_1^i[G_2^i[..[G_{n_i}^i[S_i]]..]]$$

$$R_{i+1} = G_1^i[G_2^i[..[G_{n_i}^i[S_1']]..]]$$

We abbreviate $G_1^i[..[G_{n_i}^i[..]]..]$ as $G_1^i..G_{n_i}^i[..]$. Since $R_i{\uparrow_o}$ holds for any i, there is a largest k_i, called the **nesting level** of R_i, with $1 \leq k_i \leq n_i$ such that

$$R_i = G_1^i..G_{k_i}^i[X_i] \;\text{ where } X_i{\uparrow_o}$$

The next step consists in proving an auxiliary property, namely that

$(*)$ If $X_i{\uparrow_o}$ according to **A** then $R_i{\uparrow_o}$ has an infinite quasi-lazy reduction

We show $(*)$ by induction on k_i. If $k_i = 0$ then $R_i{\uparrow_o}$ according to **A**, so it has an infinite quasi-lazy reduction. For $k_i \geq 1$, by induction hypothesis, $G_2..G_{k_i}[X_i]$ has an infinite quasi-lazy reduction, say $G_2..G_{k_i}[X_i] \rightarrow_{\beta c} Y_1 .. \rightarrow_{\beta c} Y_j \rightarrow_{\beta c} \ldots$ If $G_1 = (\lambda x.(\mathbf{C}x)L)[]$, the following reduction is infinite and quasi-lazy :

$$R_i = (\lambda x.(\mathbf{C}x)L)(G_2..G_{k_i}[X_i]) \rightarrow_c (\mathbf{C}G_2..G_{k_i}[X_i])(L[G_2..G_{k_i}[X_i]/x]) \rightarrow_{\beta c}$$

$$(\mathbf{C}Y_1)(L[G_2..G_{k_i}[X_i]/x]) \ldots \rightarrow_{\beta c} (\mathbf{C}Y_j)(L[G_2..G_{k_i}[X_i]/x]) \rightarrow_{\beta c} \ldots$$

Otherwise $G_1 = []L$, in which case the next one is an infinite quasi-lazy reduction

$$R_i = (G_2..G_{k_i}[X_i])L \rightarrow_{\beta c} (Y_1 L) \ldots \rightarrow_{\beta c} (Y_j L) \rightarrow_{\beta c} \ldots$$

The reduction from M can be of two forms:

1. $\forall i \,.\, k_i = 0$. That means, $\forall i \,.\, R_i{\uparrow_o}$ according to **A**. We will show it for $i = 0$. Suppose $R_0{\uparrow_o}$ according to **B** or **C**. Then, for some finite N, either $R_N = D_{xL}[L']$ or $R_N = E_L[L']$, with $L'{\uparrow_o}$. So $k_N \geq 1$ which contradicts the hypothesis. Therefore, using $(*)$, $F[R_0]$ has an infinite quasi-lazy reduction. So M does.

2. $\exists i \,.\, k_i > 0$. Define $< m_0, \ldots m_i, .. >$ to be a sequence of subscripts verifying:

   - m_0 is the least j for which $k_j > 0$
   - $\forall i \,.\, k_{m_i} < k_{m_{i+1}}$ & $\forall z \in [m_i, m_{i+1} - 1] \,.\, k_z = k_{m_i}$

   Hereafter, p_i stands for k_{m_i} and we forget about superscripts in contexts G. We distinguish two cases:

   (a) The sequence of indexes associated to M is finite. That is, there is a largest finite N such that $\forall j > p_N \,.\, k_j = p_N$. In other words, $\forall j > p_N \,.\, R_j = G_1..G_{p_N}[X_j]$. As in (1), it is easy to see that $X_{m_N}{\uparrow_o}$ according to **A**, for otherwise the nesting level of R_{m_N} would be greater than p_N. Therefore, property $(*)$ says that $FG_1..G_{p_N}[X_{m_N}]$ has an infinite quasi-lazy reduction. So M does.

   (b) The sequence of indexes associated to M is infinite. That is,

   $$M \xrightarrow{\star}_o F[G_1..G_{p_0}[X_{m_0}]] \xrightarrow{\star}_o F[G_1..G_{p_0}G_{p_0+1}..G_{p_1}[X_{m_1}]] ... \xrightarrow{\star}_o$$

   $$F[G_1..G_{p_0}..G_{p_i}G_{p_i+1}..G_{p_{i+1}}[X_{m_{i+1}}]] \xrightarrow{\star}_o ...$$

   where $X_{m_i}{\uparrow_o}$ and $X_{m_i} \xrightarrow{+}_o G_{p_i+1}..G_{p_{i+1}}[X_{m_{i+1}}]$ for every i.

   Note that reductions starting from X_{m_i} are non-empty. Suppose there is some j for which $X_{m_j} = G_{p_j+1}..G_{p_{j+1}}[X_{m_{j+1}}]$. Then $R_{m_j} = G_1..G_{p_j}G_{p_j+1}..G_{p_{j+1}}[X_{m_{j+1}}]$ and since $p_{j+1} > p_j$ this means that the nesting level of R_{m_j} is greater than $p_j = k_{m_j}$ which is absurd.

   We will built up an infinite quasi-lazy reduction starting from M of the form

   $$\text{(IQL)} \qquad M \xrightarrow{\star}_o Q_0[X_0] \ldots \xrightarrow{\star}_{\beta c} Q_0..Q_i[X_i] \xrightarrow{\star}_{\beta c} ...$$

   for lazy contexts $Q_0, \ldots Q_i ....$ To this end, define $l = k_{i+1} - k_i$ and $H_1 = G_{k_i+1} \ldots H_l = G_{k_{i+1}}$.

   The following holds by a straightforward induction on l:

   $$(**) \quad H_1 \ldots H_l[X] \xrightarrow{\star}_c Q[X]$$

   We sketch the inductive case. Suppose $l > 1$ and Q' is the lazy context associated to $H_2 \ldots H_l[X]$

   - either

     $$H_1 \ldots H_l[X] = (\lambda y.(\mathbf{C}y)N)[])H_2 \ldots H_l[X] \rightarrow_c (\mathbf{C}(H_2 \ldots H_l[X]))(N[(H_2 \ldots H_l[X])/y]),$$

     in which case, $Q = (\mathbf{C}Q')(N[(H_2 \ldots H_l[X])/y])$ is a lazy context,
   - or $H_1 \ldots H_l[X] = (H_2 \ldots H_l[X])N$. so $Q = (Q'N)$ is a lazy context, which finishes the proof.

   We show, by case analysis on the shape of X_{m_i}, that, for some lazy context Q_{i+1}, $X_{m_i} \xrightarrow{\star}_{\beta c} Q_{i+1}$ with at least one lazy reduction step. Hence, (IQL) is an infinite quasi-lazy reduction just by remark 7.3.

(a) If $X_{m_i} = (\lambda x.(\mathbf{C}x)L)K$ with K a value, the determinacy of $\rightarrow_o$ implies,

$$X_{m_i} \stackrel{(2)}{\rightarrow_o} L[K/x] \stackrel{\star}{\rightarrow_o} G_{p_i+1}..G_{p_{i+1}}[X_{m_{i+1}}]$$

Hence, using proposition 7.4 and property (**), one has the following reduction, with at least three lazy reduction steps:

$$X_{m_i} \stackrel{(\beta)}{\rightarrow_c} (\mathbf{C}K)(L[K/x]) \stackrel{(C2)}{\rightarrow_c} I(L[K/x]) \stackrel{(\beta)}{\rightarrow_c} L[K/x] \stackrel{\star}{\rightarrow_{\beta c}}$$

$$G_{p_i+1}..G_{p_{i+1}}[X_{m_{i+1}}] \stackrel{\star}{\rightarrow_c} Q_{i+1}$$

(b) If $X_{m_i} = (\lambda x.L)N$ with $L \neq (\mathbf{C}x)L'$, a reasoning similar to that of (a) yields a reduction from X_{m_i} having in this case at least one lazy step.

(c) If $X_{m_i} = (\lambda x.(\mathbf{C}x)L)N$ with N different from a value then the following holds, with K the value of N:

$$X_{m_i} \stackrel{\star}{\rightarrow_o} (\lambda x.(\mathbf{C}x)L)K \stackrel{+}{\rightarrow_o} G_{p_i+1}..G_{p_{i+1}}[X_{m_{i+1}}]$$

Therefore, case (a) applies. Note that N cannot diverge for otherwise X_i would be N.

(d) If $X_{m_i} = (N_1 N_2)$ with N_1 different from a value then it must be N_1 for otherwise X_{m_i} would be N_1. Thus, for K the value of N_1, one has

$$X_{m_i} \stackrel{\star}{\rightarrow_o} (K N_2) \stackrel{+}{\rightarrow_o} G_{p_i+1}..G_{p_{i+1}}[X_{m_{i+1}}]$$

The syntactic shape of (KN) corresponds either to case (a) or to case (b) so the reasoning given there is applicable.

□

Proof of proposition 2.2(2): Let M be a closed term of Λ_v and suppose $M \Downarrow_v K$. Then $M \stackrel{\star}{\rightarrow_v} K$ and, by proposition 7.6, we have $(M)^c_v \stackrel{\star}{\rightarrow_o} (K)^c_v$. Using 7.5, $(M)^c_v \stackrel{\star}{\rightarrow_{\beta c}} (K)^c_v$. So, $(M)^c_v \Downarrow_c$ by the standardization theorem for $\rightarrow_{\beta c}$ [15].

Suppose that $M \Uparrow_v$. Therefore, $M \uparrow_o$ and, by lemma 7.8, M has an infinite quasi-lazy reduction. Applying 7.4 we conclude that $M \Uparrow_c$. □

Appendix B

In this appendix we give some indications about the proofs of **B1** to **B5**, stated in section 5. Two auxiliary bisimulation equivalences are needed:

C1 $!P \sim !P|!P$

C2 $(\nu x)(P|!z(w)(Q|[\![x := H]\!])|[\![x := H]\!]) \sim (\nu x)(P|[x := H)|(\nu x)!z(w)(Q|[\![x := H]\!])$ if P and Q contain x only as an output name.

The relation $\mathcal{R}$ defined as follows is a bisimulation up to $\sim$ (using **C1**):

- $(\ (\nu x)(P|!z(w)(Q|[\![x := H]\!])|[\![x := H]\!])\ ,\ (\nu x)(P|[x := H)|(\nu x)!z(w)(Q|[\![x := H]\!])\ ) \in \mathcal{R}$, if P, Q satisfy the conditions required above.
- $(A, B) \in \mathcal{R}_1 \Rightarrow (A, B) \in \mathcal{R}$, where $\mathcal{R}_1$ is the bisimulation used in proving B1.
- $(A, B) \in \mathcal{R} \Rightarrow ((\nu y)A\ ,\ (\nu y)B) \in \mathcal{R}$.
- $(A, B) \in \mathcal{R} \Rightarrow (U|A\ ,\ U|B) \in \mathcal{R}$ for every U.

- $(A, B) \in \mathcal{R}$ and $(B, C) \in \mathcal{R} \Rightarrow (A, C) \in \mathcal{R}$.

Therefore, we have

B1 Shown in [13].

B2,B3 Direct proofs.

B4 The relation $\mathcal{R}$ built out of the pairs

- $((\nu x)(!v(z)P \mid !x(w)F) , (\nu x)!v(z')(P[z'/z] \mid !x(w)F))$
- $((\nu x)(S \mid !x(w)F \mid !v(z)P \mid !x(w)F) , (\nu x)(S \mid !x(w)F \mid !v(z')(P[z'/z] \mid !x(w)F)))$

where $v \neq x$ and z' is a new name is a bisimulation up to $\sim$ (using C1).

B5 The relation $\mathcal{R}$ whose pairs are of the form

- $((\nu x)!z(w)([\![N]\!]w|[\![x := H]\!]) , !(\nu x)z(w)([\![N]\!]w|[\![x := H]\!])$
- $((Q|(\nu x)!z(w)([\![N]\!]w|[\![x := H]\!])) , (Q|!(\nu x)z(w)([\![N]\!]w|[\![x := H]\!]))$

is a bisimulation up to $\sim$ (using C2).