On asynchrony in name-passing calculi

Massimo Merro* Davide Sangiorgi**

INRIA Sophia-Antipolis, France

Abstract. The asynchronous π-calculus is considered the basis of experimental programming languages (or proposal of programming languages) like Pict, Join, and Blue calculus. However, at a closer inspection, these languages are based on an even simpler calculus, called *Local* π (Lπ), where: (a) only the *output capability* of names may be transmitted; (b) there is no *matching* or similar constructs for testing equality between names.

We study the basic operational and algebraic theory of Lπ. We focus on bisimulation-based behavioural equivalences, precisely on *barbed congruence*. We prove two coinductive characterisations of barbed congruence in Lπ, and some basic algebraic laws. We then show applications of this theory, including: the derivability of *delayed input*; the correctness of an optimisation of the encoding of call-by-name λ-calculus; the validity of some laws for Join.

1 Introduction

The *asynchronous π-calculus* (π_{a}) is a variant of the π-calculus where message emission is non-blocking. Formally, the output prefix $\overline{a}b.\,P$ of π-calculus is replaced with the simpler output particle $\overline{a}b$, which has no continuation. The asynchronous π-calculus has been first introduced by Honda and Tokoro [17], who showed that it is expressive enough to encode the (synchronous) π-calculus.

Asynchronous communications are interesting from the point of view of concurrent and distributed programming languages, because they are easier to implement and they are closer to the communication primitives offered by available distributed systems (see also Agha's actor model [2]). The asynchronous π-calculus is considered the basis of experimental programming languages (or proposal of programming languages) like Pict [33], Join [13], π_1 [3], and the Blue calculus [11]. However, at a closer inspection, these languages are based on an even simpler calculus, where:

(a) the recipient of a name may only use it in output actions; that is, only the *output capability* of names may be transmitted;
(b) there is no *matching* construct (or similar constructs like mismatching) for testing equality between names. (We may also view (b) as a consequence of (a), since testing the identity of a name requires more than the output capability.)

---

[*] Email: `mmerro@sophia.inria.fr`.

[**] Email: `davides@sophia.inria.fr`.

These restrictions are explicit in Join and in recent proposals of the Blue calculus. In Pict, (b) is explicit; (a) is not, but most Pict programs obey it.

We call *Local* π (Lπ) the asynchronous π-calculus with the additional simplifications (a) and (b). In this paper, we study the basic operational and algebraic theory of Lπ. We focus on bisimulation-based behavioural equivalences, precisely on *barbed congruence* [28]. This equates processes that, very roughly, in all contexts give rise to the same patterns of interactions. Like other contextually-defined forms of bisimulation, barbed congruence is sensitive to the set of operators of a calculus. Lπ is a subcalculus of π_{a} and π-calculus, and therefore has a smaller number of contexts. This allows us to gain useful process equalities, as discussed below.

Proof techniques for Lπ can be exploited to reason about languages such as Pict, Join, Blue and π_1, either by directly adapting the techniques to these languages, or by means of encodings into Lπ. The theory of Lπ should also be useful for giving the semantics to, and reasoning about, concurrent or distributed object-oriented languages. For instance, (a) can guarantee the fundamental property that an object has unique identity. In an object world, the name a of an object may be transmitted; the recipient may use a to access its methods, but he/she cannot create a new object called a. When representing objects in the π-calculus, this usually translates into the constraint that the recipient of a may only use it in output. Indeed, Lπ may also be seen as a simple calculus of objects. A restriction $\boldsymbol{\nu}(a)P$ declares a new object called a. Constraint (a) ensures that all inputs at a are in P and can be statically detected. We may also say that restriction $\boldsymbol{\nu}(a)$ defines the location of a, and see Lπ as a simple calculus of distributed objects.

Studies of bisimulation-based behavioural equivalences for asynchronous mobile calculi are [17, 19, 15, 4]. In these theories, the most important algebraic law that is not in the theory of the synchronous π-calculus is

$$!a(x).\,\overline{a}x \;=\; \mathbf{0}$$

Although this law is useful (it is used for instance by Pierce and Nestmann to prove the correctness of an encoding of guarded sum [29]), it seems fair to say the restriction to asynchronous contexts does not affect much barbed congruence.

By contrast, asynchrony has strong semantic consequences under of simplifications (a) and (b). Consider these laws:

$$\overline{a}b \;=\; \boldsymbol{\nu}(c)(\overline{a}c \mid c \triangleright b), \quad \text{where } c \triangleright b \overset{\text{def}}{=} !c(x).\,\overline{b}x \text{ and } c \neq b \tag{1}$$

$$\overline{a}b \mid c \triangleright b \mid b \triangleright c \;=\; \overline{a}c \mid c \triangleright b \mid b \triangleright c \tag{2}$$

$$\boldsymbol{\nu}(c)(\overline{a}c \mid c(x)) \;=\; \boldsymbol{\nu}(c)(\overline{a}c) \;=\; \boldsymbol{\nu}(c)(\overline{a}c \mid \overline{c}b) \tag{3}$$

$$\boldsymbol{\nu}(a)(!a(x).\,R \mid (P \mid Q)) = \boldsymbol{\nu}(a)(!a(x).\,R \mid P) \mid \boldsymbol{\nu}(a)(!a(x).\,R \mid Q) \tag{4}$$

$$\text{where } a \text{ does not appear free in input position in } P, Q \text{ and } R$$

These laws are valid in Lπ, but are false in π_{a} and in π-calculus. Laws 1 and 2 are false because they equate processes that may perform syntactically different outputs: in law 1 the process on the left makes the output of a global name, whereas that on the right the output of a local (i.e., private) name; in law 2 the two processes emit two different global names. In laws 3, the derivatives of the processes after the initial output are very different, and this difference is observable in π_{a} and in π-calculus. Law 4 is a distributivity law for replicated resources (a stronger[1] version of one of Milner's replication theorems [22]).

The main inconvenience of barbed congruence is that it uses quantification over contexts in the definition, and this can make proofs of process equalities heavy. Against this, it is important to find *direct* characterisations, without context quantification. In the synchronous π-calculus barbed congruence coincides with the closure under substitutions of *early* bisimilarity; in the asynchronous π-calculus it coincides with the closure under substitutions of *asynchronous early* bisimilarity (on image-finite processes) [4, 34].

We prove two characterisations of barbed congruence in Lπ (as usual, on image-finite processes). The first is based on an embedding of Lπ into a subcalculus where all names emitted are private. Barbed congruence between processes of Lπ coincides, on their images, with a variant of asynchronous early bisimulation. The second characterisation is based on a new labeled transition system (LTS). It modifies the standard LTS so to reveal what is observable in Lπ, that is, what an external observer that behaves like a Lπ process can see by interacting with a Lπ process. Barbed congruence in Lπ coincides with the standard asynchronous early bisimulation defined on the new LTS. The resulting coinductive proof method can be enhanced by means of "bisimulation up–to" techniques.

Technical differences of these characterisations of barbed congruence in Lπ w.r.t. those in π_{a} and π-calculus are: (i) the labeled bisimulations of Lπ are congruence relations and therefore do not have to be closed under substitutions to obtain barbed congruence; (ii) in Lπ the labeled bisimulations coincide with their ground versions (which has no universal quantification on the received names); (iii) the characterisations in Lπ are proved without the matching construct (which is essential in the proofs in π_{a} and π-calculus).

The theory of Lπ (for instance, its algebraic properties and labeled bisimulations) is also useful in calculi where the usage of some names goes beyond the syntax of Lπ. For instance, there could be synchronous names, or names that can be tested for identity. A type system could be used to distinguish between "Lπ names" and the other names, and the theory of Lπ can then be applied to the former names. For simplicity we develop the theory for a monadic calculus; the generalisation to the polyadic version is straightforward.

In Section 6 we discuss some applications of the theory of Lπ. (i) We prove that in Lπ the *delayed input* (a form of non-blocking input prefixing) is derivable, and present some of its algebraic properties. (ii) We prove an optimisation of the encoding of call-by-name λ-calculus and, exploiting delayed input, we derive an encoding of *strong* call-by-name. (iii) We prove some laws for the Join-calculus. (iv) We prove some non-full abstraction and full abstraction results for the encoding used by Boreale [7] to compare the expressiveness of asynchronous

---

[1] In Milner's original theorems name a may not be exported.

mobility and internal mobility (where only private names may be passed). (v) We prove that Thielecke's axiomatic semantics of the Continuation Passing Style calculus [39] is operationally sound.

Calculi similar to Lπ are discussed in [18, 7, 3, 41]. Some of the techniques we use in Section 4 are inspired by techniques in [38]. Characterisations of barbed congruence on calculi for mobile processes include [4, 8, 3]. However, in these bisimilarity, matching transitions of processes have the same labels, therefore the problems given by restrictions (a) and (b) do not appear. Other studies of barbed congruence, or similar contextual-based bisimulations, for mobile processes include [19, 20, 42, 14, 16] (and, for a coordination language, [12]). [9] studies barbed congruence in synchronous π-calculus with capability types and no matching, of which (a) and (b) are a special case. Our characterisations are simpler than those in [9], but the latter are more general, in that they can be applied to several π-calculus languages (although the extension to asynchronous languages is not straightforward). The technical approaches are different: in [9] bisimulations have a type environment (in fact, closures) whereas in this paper bisimulations are directly defined on processes.

In this extended abstract, proofs are omitted or just sketched, so to leave space for the examples. Complete proofs can be found in [21]

2 The calculus Lπ

The grammar of Lπ has operators of inaction, input prefix, asynchronous output, parallel composition, restriction and replicated input:

$$P ::= \mathbf{0} \;\Big|\; a(x).\,P \;\Big|\; \overline{a}b \;\Big|\; P \mid P \;\Big|\; \boldsymbol{\nu}(a)P \;\Big|\; !a(x).\,P$$

with the constraint that in an input $a(x).\,P$ name x may not occur free in P in input position (this constraint shows that only the output capability of names may be transmitted).

There is no summation operator. This because in an asynchronous calculus the meaning of summation, other than input guarded, is unclear [4]. Pierce and Nestmann have showed that input-guarded summation can be coded up using asynchronous communications (their encoding respects restrictions (a) and (b)).

We use small letters $a, b, \ldots, x, y$ for *names*; capital letters P, Q, R for *processes*, σ for substitutions; $P\sigma$ is the result of applying σ to P, with the usual renaming convention to avoid captures. Parallel composition has the lowest precedence among the operators. The labeled transition system is the usual one (in the late style, [27, 38]). Transition are of the form $P \xrightarrow{\mu} P'$, where *action* μ can be: τ (interaction), $a(b)$ (input), $\overline{a}b$ (free output) and $\boldsymbol{\nu}(b)\overline{a}b$ (bound output, that is the emission of a private name b at a). In these actions, a is the *subject* and b the *object*. Free and bound names (fn, bn) of actions and processes are defined as usual. Relation $\Longrightarrow$ is the reflexive and transitive closure of $\xrightarrow{\tau}$; moreover, $\stackrel{\mu}{\Longrightarrow}$ stands for $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$, and $\stackrel{\widehat{\mu}}{\Longrightarrow}$ for $\stackrel{\mu}{\Longrightarrow}$ if $\mu \neq \tau$, and for $\Longrightarrow$ if $\mu = \tau$.

2.1. Links A *link* process behaves as a name buffer receiving names at one end-point and retransmitting them at the other end-point (in the π-calculus literature, links are sometimes called forwarders [19]). Given two names a and b, we call *static link* the process $!a(x).\,\overline{b}x$, abbreviated $a \triangleright b$.

We sometimes use a more sophisticated form of link, which does not perform free outputs: the name sent at b is not x, but a link to x (this is the definition of links in calculi where all outputs are bound [37]). We call this a *dynamic link* process, written $a \rightarrow b$, and defined using recursion thus

$$a \rightarrow b \overset{\text{def}}{=} !a(x).\,\boldsymbol{\nu}(c)(\overline{b}c \mid c \rightarrow x).$$

Remark 1. The dynamic link process $a \rightarrow b$ is not in Lπ, but it is synchronous early bisimilar (Definition 2) to a process of Lπ (using replication in place of recursion).

3 Some background on barbed congruence

Below, we define barbed congruence on a generic subset $\mathcal{L}$ of π-calculus processes (transitions between π-calculus processes are of the same form as for Lπ processes). A *$\mathcal{L}$-context* is a process of $\mathcal{L}$ with a single hole $[\cdot]$ in it. We write $P \downarrow_a$ if P can make an output action whose subject is a, that is if there exist P', b such that $P \xrightarrow{\overline{a}b} P'$ or $P \xrightarrow{\boldsymbol{\nu}(b)\overline{a}b} P'$. We write $P \Downarrow_a$ if $P \Longrightarrow P'$ and $P' \downarrow_a$.

Definition 1 (barbed bisimulation and congruence).[2] *A symmetric relation S on π-calculus processes is a* barbed bisimulation *if $P \mathrel{\mathcal{S}} Q$ implies:*

1. *If $P \xrightarrow{\tau} P'$ then $\exists\, Q'$ s.t. $Q \Longrightarrow Q'$ and $P' \mathrel{\mathcal{S}} Q'$.*
2. *If $P \downarrow_a$ then $Q \Downarrow_a$.*

Two π-calculus processes P and Q are barbed bisimilar*, written $P \stackrel{\centerdot}{\approx} Q$, if $P \mathrel{\mathcal{S}} Q$ for some barbed bisimulation $\mathcal{S}$. Let $\mathcal{L}$ be a set of π-calculus processes, and $P, Q \in \mathcal{L}$. We say that P and Q are* barbed congruent in $\mathcal{L}$*, written $P \approx^{\mathcal{L}}_{bc} Q$, if for each $\mathcal{L}$-context $C[\cdot]$, it holds that $C[P] \stackrel{\centerdot}{\approx} C[Q]$.*

In CCS barbed congruence coincides with observation congruence. In π_{a} with matching operator, barbed congruence coincides with the closure under substitutions of *asynchronous early bisimulation* [34, 4]. Similarly, in π-calculus with matching operator, it coincides with the closure under substitutions of *synchronous early bisimulation.* These two bisimulations only differ in the input clause.

Definition 2 (early bisimulations). *A symmetric relation $\mathcal{S}$ on π-terms is an $o\tau$-bisimulation if $P \mathrel{\mathcal{S}} Q$, $P \xrightarrow{\mu} P'$, μ is not an input and* $\mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$, *implies that there exists Q' s.t. $Q \overset{\widehat{\mu}}{\Longrightarrow} Q'$ and $P' \mathrel{\mathcal{S}} Q'$.*

---

[2] In π-calculus, the observability predicate normally checks also the possibility of input actions; observing only outputs does not affect the resulting barbed congruence.

- synchronous early bisimulation *is the largest* $o\tau$*-bisimulation* $\mathcal{S}$ *on* π*-calculus s.t.* $P \mathrel{\mathcal{S}} Q$ *and* $P \xrightarrow{a(x)} P'$ *implies that* $\forall\, b,\ \exists\, Q'$ *s.t.* $Q \overset{a(x)}{\Longrightarrow} Q'$ *and* $P'\{b/x\} \mathrel{\mathcal{S}} Q'\{b/x\}$.
- asynchronous early bisimulation *is the largest* $o\tau$*-bisimulation* $\mathcal{S}$ *on* π_{a} *processes s.t.* $P \mathrel{\mathcal{S}} Q$ *and* $P \xrightarrow{a(x)} P'$ *implies that* $\forall\, b,\ \exists\, Q'$ *s.t.:*
  1. *either* $Q \overset{a(x)}{\Longrightarrow} Q'$ *and* $P'\{b/x\} \mathrel{\mathcal{S}} Q'\{b/x\}$
  2. *or* $Q \Longrightarrow Q'$ *and* $P'\{b/x\} \mathrel{\mathcal{S}} (Q' \mid \overline{a}b)$.

The proofs of the above-mentioned characterisations are usually given on the class of the image finite processes (to which most of the processes one would like to write belongs) by exploiting the n-approximants of the labeled equivalences. We recall that the class of *image-finite processes* is the largest subset $\mathcal{I}$ of π-calculus process which is derivation closed and s.t. $P \in \mathcal{I}$ implies that, for all μ, the set $\{P' \ :\ P \overset{\mu}{\Longrightarrow} P'\}$, quotiented by alpha conversion, is finite. In the proofs of these characterisations, a central role is played by the *matching* construct. If matching is removed from the language, then (the closure under substitutions of) early bisimulation still implies barbed congruence, but the vice versa is false.

In the asynchronous π-calculus without matching, asynchronous early bisimulation coincides with its induced congruence and also with asynchronous *ground* bisimulation, which differs from the early one in that it has no universal quantification in the input clause.

Definition 3 (Asynchronous ground bisimulation). *The* asynchronous ground bisimulation *is the largest* $o\tau$*-bisimulation* $\mathcal{S}$ *on* π_{a} *processes s.t.* $P \mathrel{\mathcal{S}} Q$ *and* $P \xrightarrow{a(x)} P'$ *implies that* $\exists\, Q'$ *s.t.:*

1. *either* $Q \overset{a(x)}{\Longrightarrow} Q'$ *and* $P' \mathrel{\mathcal{S}} Q'$
2. *or* $Q \Longrightarrow Q'$ *and* $P' \mathrel{\mathcal{S}} (Q' \mid \overline{a}x)$.

Remark 2. Also for the labeled bisimulations we shall study for Lπ, the early and the ground versions coincide. For this reason, in Lπ we shall simply present the ground versions.

4 Eliminating free output transitions

In this section we prove a characterisation for barbed congruence in Lπ by exploiting a compositional encoding $[\![\ ]\!]$ (essentially Boreale's [7]), which is an homomorphism on all operators except output, for which we have:

$$[\![\overline{a}b]\!] \overset{\text{def}}{=} \overline{a}[b] \quad \text{where } \ \overline{a}[b] \overset{\text{def}}{=} \boldsymbol{\nu}(d)(\overline{a}d \mid d \rightarrow b) \ \text{ with } \ d \notin \{a, b\}$$

Remark 3. The process $\overline{a}[b]$ is not in Lπ, but, by Remark 1, it is synchronous early bisimilar (Definition 2) to a process of Lπ.

Let $\approx_{\mathrm{L}}$ be the variant of asynchronous ground bisimulation in which the output $\overline{a}x$ is replaced by $\overline{a}[x]$ (clause 2, Definition 3). The proof technique for proving that two processes of $\mathrm{L}\pi$ are (or are not) barbed congruent consists in translating them, and then checking that their images are (or are not) in the relation $\approx_{\mathrm{L}}$.

Lemma 1 (Boreale). *Let P and Q be two processes in $\mathrm{L}\pi$. Then*

$$P \stackrel{\centerdot}{\approx} Q \text{ iff } [\![P]\!] \stackrel{\centerdot}{\approx} [\![Q]\!].$$

Theorem 1 (First characterisation of barbed congruence in $\mathrm{L}\pi$). *Let P and Q be two processes in $\mathrm{L}\pi$. Then*

1. $P \approx^{\mathrm{L}\pi}_{\mathrm{bc}} Q$ implies $[\![P]\!] \approx_{\mathrm{L}} [\![Q]\!]$, *for P and Q image finite processes*
2. $[\![P]\!] \approx_{\mathrm{L}} [\![Q]\!]$ implies $P \approx^{\mathrm{L}\pi}_{\mathrm{bc}} Q$.

Proof.

1. We prove that $[\![P]\!] \approx_{\mathrm{L}} [\![Q]\!]$ when for each $R \in \mathrm{L}\pi$ $[\![P \,|\, R]\!] \stackrel{\centerdot}{\approx} [\![Q \,|\, R]\!]$ holds. By Lemma 1 we can conclude.
2. By proving that for each context $C[\cdot]$ in $\mathrm{L}\pi$ $[\![C[P]]\!] \approx_{\mathrm{L}} [\![C[Q]]\!]$. This implies $[\![C[P]]\!] \stackrel{\centerdot}{\approx} [\![C[Q]]\!]$ and therefore, by Lemma 1, $C[P] \stackrel{\centerdot}{\approx} C[Q]$.

By contrast with other characterisations of barbed congruence in the literature, in the proof above we do not need matching because only private names are emitted. We use the above result in some examples of Section 6, and in the next section to derive another proof technique for $\mathrm{L}\pi$.

5 A labeled bisimulation for $\mathrm{L}\pi$

In this section we give a more powerful proof technique, in whose correctness proof, Theorem 1 is important. Table 1 gives a new Labelled Transition System (LTS) $\xmapsto{\mu}$ for $\mathrm{L}\pi$. We prove that asynchronous ground bisimulation defined on the new LTS coincides with barbed congruence in $\mathrm{L}\pi$.[3] We recall that (the closure under substitution of) asynchronous early bisimulation on the original LTS $\xrightarrow{\mu}$ coincides with barbed congruence in π_{a}. Therefore the difference between the two LTSs shows the difference between what is observable in $\mathrm{L}\pi$, and in π_{a} (or π-calculus) with matching. The new LTS is defined on top of the original one, and transforms the output of a name b into the output of a fresh *pointer* p to b. We call p a pointer to b because a link $p \triangleright b$ is introduced through which any output along p is redirected onto b. The weak transitions $\overset{\mu}{\Longmapsto}$ and $\Longmapsto$ for the new LTS are defined from the strong transitions $\xmapsto{\mu}$ and $\xmapsto{\tau}$ in the usual way. We write $\overset{\mapsto}{\approx}_{\mathrm{a}}$ to denote the relation obtained by replacing, in Definition 3, arrow $\longrightarrow$ with $\longmapsto$ and arrow $\Longrightarrow$ with $\Longmapsto$.

---

[3] It also coincides with the early version, see Remark 2

$$\texttt{free-out:}\ \frac{P \xrightarrow{\overline{a}b} P' \qquad p \notin \mathrm{fn}(P)}{P \overset{\boldsymbol{\nu}(p)\overline{a}p}{\longmapsto} (p \triangleright b \mid P')} \qquad \texttt{bound-out:}\ \frac{P \xrightarrow{\boldsymbol{\nu}(b)\overline{a}b} P' \qquad p \notin \mathrm{fn}(P)}{P \overset{\boldsymbol{\nu}(p)\overline{a}p}{\longmapsto} \boldsymbol{\nu}(b)(p \triangleright b \mid P')}$$

$$\texttt{sync:}\ \frac{P \xrightarrow{\tau} P'}{P \overset{\tau}{\longmapsto} P'} \qquad \texttt{input:}\ \frac{P \xrightarrow{a(b)} P'}{P \overset{a(b)}{\longmapsto} P'}$$

Table 1. A new labeled transition system for Lπ

Lemma 2. *Let P and Q be two processes in* Lπ. *Then*

$$P \overset{\mapsto}{\approx}_{\mathrm{a}} Q \ \text{ iff } \ [\![P]\!] \approx_{\mathrm{L}} [\![Q]\!].$$

Theorem 2 (Second characterisation of barbed congruence in Lπ). *Let P and Q be two processes in* Lπ. *Then*

1. $P \approx_{\mathrm{bc}}^{\mathrm{L}\pi} Q$ implies $P \overset{\mapsto}{\approx}_{\mathrm{a}} Q$, *for P and Q image finite processes*
2. $P \overset{\mapsto}{\approx}_{\mathrm{a}} Q$ implies $P \approx_{\mathrm{bc}}^{\mathrm{L}\pi} Q$.

Proof. By Lemma 2 and Theorem 1.

Both the characterisation of barbed congruence in Lπ in Theorem 1 and the characterisation above are based on the use of links. In the former characterisation, links are added statically via an encoding (at "compile-time"); in the latter characterisation, they are added dynamically in the bisimulation game (at "run-time"). The advantage of the latter characterisation is that: (i) it uses simpler links $p \triangleright b$ instead of links $p \rightarrow b$; (ii) links are not added in case of internal communications; (iii) in the input clause, it uses the particle $\overline{a}x$ instead of $\overline{a}[x]$ (that produces links). An even more important advantage of the latter characterisation is that the number of the added links may be further reduced using bisimulation up-to context and up-to *expansion* techniniques [36] (the expansion relation is an asymmetric variant of the synchronous early bisimulation in Definition 2). For instance, under certain hypotheses on the occurrencies of b in P, the process $\boldsymbol{\nu}(b)(p \triangleright b \mid P')$ can be replaced by $P'\{p/b\}$ and the link added in rule `bound-out` can be removed. Similarly, it is easy to prove that $P \overset{\mapsto}{\approx}_{\mathrm{a}} Q$ holds when $p \triangleright b | P \overset{\mapsto}{\approx}_{\mathrm{a}} p \triangleright b | Q$ and $p \notin \mathrm{fn}(P \mid Q)$.

6 Applications

We report some applications of the theory of Lπ; the results we give fail in π_{a} and π-calculus. Further examples are reported in the full paper [21]: for instance, other replication theorems in addition to that of law 4.

6.1. Some laws

Using either Theorem 1(2) or Theorem 2(2) it is simple to prove laws 1-4 in the introduction. Law 1 is a special case of the following law, where c is not free in P in input position and $b \neq c$:

$$P\{b/c\} = \boldsymbol{\nu}(c)(P \mid c \triangleright b) \tag{5}$$

$$\texttt{del-i:}\ \frac{}{a(b)P \xrightarrow{a(b)} P} \qquad \texttt{close:}\ \frac{P \xrightarrow{\sigma_b \overline{a}b} P' \quad Q \xrightarrow{a(b)} Q' \quad \mathrm{bn}(\sigma_b) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\tau} \sigma_b(P' \mid Q')}$$

$$\texttt{self-com1:}\ \frac{P \xrightarrow{\overline{a}c} P'}{a(b)P \xrightarrow{\tau} \boldsymbol{\nu}(b)(P'\{c/b\})} \qquad \texttt{self-com2:}\ \frac{P \xrightarrow{\sigma_c \overline{a}c} P' \qquad b \notin \mathrm{n}(\sigma_c \overline{a}c)}{a(b)P \xrightarrow{\tau} \sigma_c(P'\{c/b\})}$$

$$\texttt{pass-i:}\ \frac{P \xrightarrow{\mu} P' \quad b \notin \mathrm{n}(\mu) \quad a \notin \mathrm{bn}(\mu)}{a(b)P \xrightarrow{\mu} a(b)P'} \qquad \texttt{open-i:}\ \frac{P \xrightarrow{\overline{a}b} P' \qquad b \neq a}{c(b)P \xrightarrow{c(b)\overline{a}b} P'}$$

$$\texttt{pass-}\boldsymbol{\nu}\texttt{:}\ \frac{P \xrightarrow{\mu} P' \quad b \notin \mathrm{n}(\mu)}{\boldsymbol{\nu}(b)P \xrightarrow{\mu} \boldsymbol{\nu}(b)P'} \qquad \texttt{open-}\boldsymbol{\nu}\texttt{:}\ \frac{P \xrightarrow{\mu} P' \quad [\mu = \overline{a}c \vee \mu = c(b)\overline{a}b] \quad a \neq c}{\boldsymbol{\nu}(c)P \xrightarrow{\boldsymbol{\nu}(c)\mu} P'}$$

Table 2. Inference rules for delayed input and restriction.

We call law 5 *eta rule*. It is valid in Lπ but not in π_{a} and π-calculus. A similar law, but with the double link $c \triangleright b \mid b \triangleright c$ in place of $c \triangleright b$, is given in [19].

6.2. The delayed input In an asynchronous calculus message emission is non-blocking. Milner, Parrow, Victor and others have advocated also non-blocking message reception (which is among the motivations for the Update and the Fusion calculi [31, 32] and, implicitely for the Chi calculus [14]). Such a *delayed input prefix*, written $a(x)P$, should allow the continuation P to evolve underneath the input guard, except for observable actions along x. The delayed input replaces *temporal precedencies*, imposed by plain input, with *causal dependencies*. This appears, for instance, in Abramsky's representation of Linear Logic proofs as π-calculus processes [1, 6]. Non-blocking message reception has been studied by Bellin and Scott [6], Boudol [10], Fu [14], Parrow and Victor [32], Yoshida [41] and van Breugel [40]. Bellin and Scott give a reduction semantics for a version of π-calculus, proposed by Milner, where both message emission and message reception are non-blocking [23]; van Breugel defines a labelled transition system for such calculus and proves a correspondence with the Bellin and Scott's reduction semantics.

Let DLπ be the calculus obtained by adding the delayed input construct $a(b)P$ to the grammar of Lπ (with the same constraint as plain input that b may not appear free in P in input position). In Table 2, we give the transition rules of delayed input in DLπ (we give also the rules of restriction because they are affected by the addition of delayed input). Our rules have two main differences w.r.t. van Breugel's rules [40]: (i) actions have a simpler syntax, because only the output capability of name may be transmitted; (ii) a restriction $\boldsymbol{\nu}(b)$ is added in rule `self-com1` to model *self communications*, as in $a(b)(\overline{a}b \mid P) \xrightarrow{\tau} \boldsymbol{\nu}(b)P$. We prove that the delayed input is a derived operator in Lπ.

Our actions are defined as follows:

$$\mu ::= \tau \;\Big|\; a(b) \;\Big|\; \overline{a}b \;\Big|\; \sigma_b \overline{a}b \quad \text{where} \quad \sigma_b ::= \boldsymbol{\nu}(b) \;\Big|\; c(b) \;\Big|\; \boldsymbol{\nu}(c)c(b)$$

σ_b represents the binding part of bound output actions. We define: $\mathrm{fn}(\boldsymbol{\nu}(b)\overline{a}b) = \{a\}$, $\mathrm{bn}(\boldsymbol{\nu}(b)\overline{a}b) = \{b\}$, $\mathrm{fn}(c(b)\overline{a}b) = \{c, a\}$, $\mathrm{bn}(c(b)\overline{a}b) = \{b\}$, $\mathrm{fn}(\boldsymbol{\nu}(c)c(b)\overline{a}b) = \{a\}$, $\mathrm{bn}(\boldsymbol{\nu}(c)c(b)\overline{a}b) = \{c, b\}$. We define an encoding $\{\!|\,|\!\}$, from DLπ to Lπ, and prove that it is fully abstract for barbed congruence. The encoding $\{\!|\,|\!\}$ is an homomorphism on all operators except delayed input:

$$\{\!| a(b)P |\!\} \overset{\mathrm{def}}{=} \boldsymbol{\nu}(b)(a(c).\, b \triangleright c \mid \{\!| P |\!\})$$

(A similar encoding, but with the double link $c \triangleright b \mid b \triangleright c$ in place of $b \triangleright c$, is suggested by Yoshida in [41].) In Lemma 3, $\approx$ is the synchronous early bisimulation of Definition 2; $\llbracket\ \rrbracket$ is the extension of the encoding of Section 4 to DLπ which is an homomorphism also on the delayed input; $\{\!|\,|\!\}_{\mathcal{D}}$ is the variant of $\{\!|\,|\!\}$ with $\{\!| a(b)P |\!\}_{\mathcal{D}} \overset{\mathrm{def}}{=} \boldsymbol{\nu}(b)(a(c).\, b \to c \mid \{\!| P |\!\}_{\mathcal{D}})$.

Lemma 3. *If $P \in \mathrm{DL}\pi$ then $\llbracket P \rrbracket \approx \{\!| \llbracket P \rrbracket |\!\}_{\mathcal{D}}$.*

Proof. $\llbracket P \rrbracket$ may perform only output actions of the form $\boldsymbol{\nu}(b)\overline{a}b$.

Theorem 3 (Full-abstraction of $\{\!|\,|\!\}$). *If $P, Q \in \mathrm{DL}\pi$ then:*

1. $P \approx_{\mathrm{bc}}^{\mathrm{DL}\pi} Q$ implies $\{\!| P |\!\} \approx_{\mathrm{bc}}^{\mathrm{L}\pi} \{\!| Q |\!\}$, *for P and Q image finite processes*
2. $\{\!| P |\!\} \approx_{\mathrm{bc}}^{\mathrm{L}\pi} \{\!| Q |\!\}$ implies $P \approx_{\mathrm{bc}}^{\mathrm{DL}\pi} Q$.

Proof.

1. $P \approx_{\mathrm{bc}}^{\mathrm{DL}\pi} Q \overset{1}{\Longrightarrow} \llbracket P \rrbracket \approx_{\mathrm{L}} \llbracket Q \rrbracket \overset{2}{\Longrightarrow} \{\!| \llbracket P \rrbracket |\!\}_{\mathcal{D}} \approx_{\mathrm{L}} \{\!| \llbracket Q \rrbracket |\!\}_{\mathcal{D}} \overset{3}{\Longrightarrow} \llbracket \{\!| P |\!\} \rrbracket \approx_{\mathrm{L}} \llbracket \{\!| Q |\!\} \rrbracket \overset{4}{\Longrightarrow} \{\!| P |\!\} \approx_{\mathrm{bc}}^{\mathrm{L}\pi} \{\!| Q |\!\}$. Step 1 uses an extension of Theorem 1(1) to DLπ processes, step 2 Lemma 3, step 3 the definition of the encodings, step 4 Theorem 1(1).
2. Similar to the previous case, using part (2) of Theorem 1.

Using $\{\!|\,|\!\}$ and the theory of Lπ we can prove laws for delayed input like:

$$a(b)(P \mid Q) = (a(b)P) \mid Q \qquad \text{if } b \notin \mathrm{fn}(Q) \tag{6}$$

$$a(b)c(d)P = c(d)a(b)P \qquad \text{if } c \neq b \text{ and } d \neq a \tag{7}$$

$$\boldsymbol{\nu}(a)(a(x)(\overline{a}x \mid P)) = \boldsymbol{\nu}(x)P \qquad \text{if } a \notin \mathrm{fn}(P) \tag{8}$$

Laws 6 and 7 are similar to laws of restriction. Law 8 transforms a delayed input binder into a restriction binder (it might be interesting to examine delayed input from within action calculi [25]; for instance, law 8 is riminiscient of the definition of restriction in reflexive action calculi [26]).

6.3. Encodings of the λ-calculus

(In this example, we use polyadicity, which is straightforward to accommodate in the theory of Lπ developed in the previous sections; we write $\overline{a}\langle b_1 \dots b_n \rangle$ for outputs.) The following is Milner's encoding of call-by-name λ-calculus into π-calculus (more precisely, the variant in [30], whose target calculus is Lπ).

$$
\begin{aligned}
\llparenthesis\, \lambda x.\, M \,\rrparenthesis_p &\overset{\text{def}}{=} \boldsymbol{\nu}(v)(\overline{p}\langle v\rangle \mid v(x,q).\, \llparenthesis\, M \,\rrparenthesis_q) \\
\llparenthesis\, x \,\rrparenthesis_p &\overset{\text{def}}{=} \overline{x}\langle p\rangle \\
\llparenthesis\, MN \,\rrparenthesis_p &\overset{\text{def}}{=} \boldsymbol{\nu}(q)\Big(\llparenthesis\, M \,\rrparenthesis_q \mid q(v).\, \boldsymbol{\nu}(x)(\overline{v}\langle x,p\rangle \mid !x(r).\, \llparenthesis\, N \,\rrparenthesis_r)\Big)
\end{aligned}
$$

This is also an encoding into (polyadic) Lπ. Using the eta law 5 we can prove the following optimisation of the definition of application in the case when the argument is a variable (a tail-call-like optimisation):

$$\llparenthesis\, My \,\rrparenthesis_p \overset{\text{def}}{=} \boldsymbol{\nu}(q)\Big(\llparenthesis\, M \,\rrparenthesis_q \mid q(v).\, \overline{v}\langle y,p\rangle\Big)$$

We can also exploit the delayed input operator, that is a derived operator in Lπ, to get an encoding of the *strong* call-by-name strategy, where reductions can also occur underneath an abstraction (i.e., the `Xi` rule, saying that if $M \longrightarrow M'$ then $\lambda x.\, M \longrightarrow \lambda x.\, M'$, is allowed). For this, we have to relax, in the translation of $\lambda x.\, M$, the sequentiality imposed by the input prefix $v(x,q)$ that guards the body $\llparenthesis\, M \,\rrparenthesis_q$ of the function. Precisely, we have to replace this input with a delayed input:

$$\llparenthesis\, \lambda x.\, M \,\rrparenthesis_p \overset{\text{def}}{=} \boldsymbol{\nu}(v)(\overline{p}\langle v\rangle \mid v(x,q)\llparenthesis\, M \,\rrparenthesis_q) \tag{9}$$

Using the above encoding of delayed input, we get:

$$\llparenthesis\, \lambda x.\, M \,\rrparenthesis_p \overset{\text{def}}{=} \boldsymbol{\nu}(v,x,q)\Big(\overline{p}\langle v\rangle \mid v(y,r).\, (x \triangleright y \mid q \triangleright r) \mid \llparenthesis\, M \,\rrparenthesis_q\Big)$$

One can prove results of operational correspondence and validity of β-reduction for this encoding similar to those in [24, 35] for the call-by-name λ-calculus. (The modelling of strong reductions is a major motivation behind Fusion and Chi; indeed both calculi allows us to encode strong call-by-name λ-calculus [32, 14]).

6.4. Some properties for the Join-calculus We apply the theory of Lπ to prove some behavioural equivalences of the Join-calculus. Fournet and Gonthier define the syntax of core Join thus [13]:

$$P ::= a\langle b\rangle \;\Big|\; P_1 \mid P_2 \;\Big|\; \texttt{def}\ a\langle x\rangle \,|\, b\langle y\rangle = P_1\ \texttt{in}\ P_2$$

A derived construct is `def` $a\langle x\rangle = P_1$ `in` P_2 (with a single pattern). To explain the syntax above and study its expressiveness, Fournet and Gonthier give this encoding of the Join-calculus into the ordinary π-calculus:

$$
\begin{aligned}
&\langle\!|\, a\langle b\rangle \,|\!\rangle \overset{\text{def}}{=} \overline{a}b \qquad\qquad \langle\!|\, P \mid Q \,|\!\rangle \overset{\text{def}}{=} \langle\!|\, P \,|\!\rangle \mid \langle\!|\, Q \,|\!\rangle \\
&\langle\!|\, \texttt{def}\ a\langle x\rangle \,|\, b\langle y\rangle = P_1\ \texttt{in}\ P_2 \,|\!\rangle \overset{\text{def}}{=} \boldsymbol{\nu}(a,b)(!a(x).\, b(y).\, \langle\!|\, P_1 \,|\!\rangle \mid \langle\!|\, P_2 \,|\!\rangle)
\end{aligned}
$$

This encoding, as an encoding of Join into π_{a} or π-calculus, is not full-abstract. To obtain full abstraction, Fournet and Gonthier have to add a layer of “firewalls” to the encoding. We believe that the above encoding is fully abstract as an encoding from Join to Lπ (a similar conjecture is made by Fournet and Gonthier [13]). It is easy to prove soundness, and this is sufficient for using the encoding and the theory of Lπ for proving properties of Join processes.

Theorem 4 (soundness of $\llparenthesis\ \rrparenthesis$). *Let P and Q be two processes of core Join. Then $\llparenthesis P \rrparenthesis \approx^{\mathrm{L}\pi}_{\mathrm{bc}} \llparenthesis Q \rrparenthesis$ implies $P \approx^{\mathrm{J}}_{\mathrm{bc}} Q$ ($\approx^{\mathrm{J}}_{\mathrm{bc}}$ is barbed congruence in core Join).*

Using this theorem and the theory of Lπ we can prove laws for the Join-calculus, for instance:

(J1) $\mathtt{def}\ a\langle x\rangle = R\ \mathtt{in}\ P \mid Q \ \approx^{\mathrm{J}}_{\mathrm{bc}}\ (\mathtt{def}\ a\langle x\rangle = R\ \mathtt{in}\ P) \mid (\mathtt{def}\ a\langle x\rangle = R\ \mathtt{in}\ Q)$
(J2) $\mathtt{def}\ a\langle x\rangle = b\langle x\rangle\ \mathtt{in}\ P \ \approx^{\mathrm{J}}_{\mathrm{bc}}\ P\{^{b}\!/\!_{a}\}$
(J3) $\mathtt{def}\ a\langle x\rangle = P\ \mathtt{in}\ C[a\langle b\rangle] \ \approx^{\mathrm{J}}_{\mathrm{bc}}\ \mathtt{def}\ a\langle x\rangle = P\ \mathtt{in}\ C[P\{^{b}\!/\!_{x}\}]$
if context $C[\cdot]$ does not capture name a.

Laws (J1) and (J2) are the Join-calculus versions of laws 4 and 5 respectively. Law (J3) reminds us of *inline expansion*, an optimization technique for functional languages which replaces a function call with a copy of the function body. An instance of law (J3) is

$$\mathtt{def}\ a\langle x\rangle = P\ \mathtt{in}\ (Q \mid a\langle b\rangle) \ \approx^{\mathrm{J}}_{\mathrm{bc}}\ \mathtt{def}\ a\langle x\rangle = P\ \mathtt{in}\ (Q \mid P\{^{b}\!/\!_{x}\})$$

that shows a sort of insensitiveness to τ-actions (the process on the right is obtained from the process on the left by performing a τ-step). None of these laws can be proved using encoding $\llparenthesis\ \rrparenthesis$ and π_{a} or π-calculus (if the local name a is exported, the encodings of the processes in the laws can be distinguished both in π_{a} and π-calculus). In [8], a labeled bisimulation for the Join calculus is introduced. However, in this bisimulation the labels of the matching transitions must be syntactically the same. Therefore laws like (J2) cannot be proved.

6.5. Full abstraction of $[\![\ ]\!]$

Sangiorgi [37] introduces a subcalculus of the π-calculus, called πI, where only private names may be emitted, that is, output processes have the form $\boldsymbol{\nu}(c)(\overline{a}c.\, P)$. In [7] Boreale uses (a slight variant of) encoding $[\![\ ]\!]$ of Section 4 to show that any process in Lπ can be compiled onto πI. Boreale leaves as an open problem whether the encoding is fully abstract for some reasonable behavioural equivalence. We can prove that Boreale's encoding *is not* fully abstract as an encoding from Lπ to πI, assuming that the behavioural equivalence for both the source and the target calculus is barbed congruence. As a counterexample, take $P = !a(x).\, \overline{a}x$, $Q = \mathbf{0}$; then $P \approx^{\mathrm{L}\pi}_{\mathrm{bc}} Q$ but not $[\![P]\!] \approx^{\pi\mathrm{I}}_{\mathrm{bc}} [\![Q]\!]$. This is not surprising because the source language is asynchronous while the target language is synchronous. However, also if we considered as target language the asynchronous variant of πI (where output processes have the form $\boldsymbol{\nu}(c)(\overline{a}c \mid P)$), the encoding *is not* fully abstract (as a conterexample, take the same processes P and Q above). We can prove that (on image finite processes) the encoding *is* fully abstract if the target calculus is the asynchronous πI where only output capability of names may be transmitted. We denote by LπI (*Local* πI) this calculus.

Theorem 5 (Full-abstraction of $[\![\ ]\!]$). *Let* P, Q *be two processes in* $\mathrm{L}\pi$. *Then*

$$P \approx^{\mathrm{L}\pi}_{\mathrm{bc}} Q \text{ iff } [\![P]\!] \approx^{\mathrm{L}\pi\mathrm{I}}_{\mathrm{bc}} [\![Q]\!].$$

Proof. As to the implication from left to right, by Theorem 2(2), we have $P \approx^{\mathrm{L}\pi}_{\mathrm{bc}} [\![P]\!]$ for each $P \in \mathrm{L}\pi$; hence $[\![P]\!] \approx^{\mathrm{L}\pi}_{\mathrm{bc}} [\![Q]\!]$. Because $\mathrm{L}\pi\mathrm{I} \subset \mathrm{L}\pi$, this implies $[\![P]\!] \approx^{\mathrm{L}\pi\mathrm{I}}_{\mathrm{bc}} [\![Q]\!]$. The implication from right to left follows by Lemma 1 and by compositionality of $[\![\ ]\!]$.

6.6. Operational soundness of CPS axioms

Thielecke studies the target calculi of Continuation Passing Style transforms [39]. For this, he introduces a CPS calculus, similar to the intermediate language of Appel's compiler [5]. The CPS calculus is very simple and low-level: only variables may be passed as arguments, moreover application is like a jump, with variables as argument. Thielecke gives an axiomatic semantics for the CPS calculus, as the congruence induced by four axioms. We have exploited an encoding of the CPS-calculus into π-calculus given by Thielecke (which is also an encoding into $\mathrm{L}\pi$) and the theory of $\mathrm{L}\pi$ to prove that the axiomatic semantics of the CPS-calculus is sound w.r.t. its operational semantics (defined using *Morris's context-equivalence*; this equivalence, on confluent calculi like the CPS calculus, coincides with barbed congruence). For this, we exploit eta rule 5, law 4 (and other distributivity laws) and the fact that bisimulation in $\mathrm{L}\pi$ is a congruence relation (these laws and properties fail in π_{a} and π-calculus).

Acknowledgements The authors were partially supported by France Télécom, CTI-CNET 95-1B-182 Modélisation de Systémes Mobiles.

We thank Gérard Boudol, Ilaria Castellani, Silvano Dal-Zilio, Matthew Hennessy, Uwe Nestmann, Benjamin Pierce and Nobuko Yoshida for stimulating and insightful discussions.

References

1. S. Abramsky. Proofs as Processes. *Theoretical Computer Science*, 135(1):5–9, December 1994.
2. G. Agha. *Actors: a Model of Concurrent Computation in Distributed Systems.* The MIT Press, 1986.
3. R. Amadio. An asynchronous model of locality, failure, and process mobility. In Proc. Coordination'97, volume 1282 of *Lecture Notes in Computer Science.* Springer Verlag, 1997.
4. R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. In *Proc. CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science.* Springer Verlag, 1996.
5. A. Appel. *Compiling with Continuations.* Cambridge University Press, 1992.
6. G. Bellin and P. Scott. On the π-calculus and Linear Logic. *Theoretical Computer Science*, 135(1):11–65, December 1994.

7. M. Boreale. On the expressiveness of internal mobility in name-passing calculi. In *Proc. CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.
8. M. Boreale, C. Fournet, and C. Laneve. Bisimulations for the Join Calculus. Proc. IFIP Conference PROCOMET'98, 1997.
9. M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. To appear in *Proc. LICS'98*, IEEE Computer Society Press., 1998.
10. G. Boudol. Some Chemical Abstract Machines. In *Proc. Rex School/Symposium 1993 "A Decade of Concurrency — Reflexions and Perspectives"*, volume 803 of *Lecture Notes in Computer Science*, pages 92–123. Springer Verlag, 1994.
11. G. Boudol. The pi-calculus in direct style. In *24th POPL*. ACM Press, 1997.
12. N. Busi, R. Gorrieri, and G. Zavattaro. A process algebraic view of linda coordination primitives. *Theoretical Computer Science*, 192(2):167–199, 1988.
13. Fournet C. and Gonthier G. The Reflexive Chemical Abstract Machine and the Join calculus. In *Proc. 23th POPL*. ACM Press, 1996.
14. Y. Fu. A proof theoretical approach to communication. In *24th ICALP*, volume 1256 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.
15. M. Hansen, H. Hüttel, and J. Kleist. Bisimulations for asynchronous mobile processes. In *Proc. Tbilisi Symposium on Language, Logic, and Computation*, 1996. Also available as BRICS Report No. EP-95-HHK, BRICS, Aalborg University, Denmark 1996.
16. M. Hennessy and J. Riely. A typed language for distributed mobile processes. In *25th POPL*. ACM Press, 1997.
17. K. Honda and M. Tokoro. An Object Calculus for Asynchronous Communications. In M. Tokoro, O. Nierstrasz, P. Wegner, and A. Yonezawa, editors, *ECOOP '91 Workshop on Object Based Concurrent Programming,* Geneva, Switzerland, 1991 , volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer Verlag, 1991.
18. K. Honda and M. Tokoro. A Small Calculus for Concurrent Objects. In *OOPS Messanger*, Association for Computing Machinery. 2(2):50-54, 1991.
19. K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.
20. N. Kobayashi, B.C. Pierce, and D.N. Turner. Linearity and the pi-calculus. In *Proc. 23th POPL*. ACM Press, 1996.
21. M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. To appear as INRIA technical report, 1998.
22. R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, LFCS, Dept. of Comp. Sci., Edinburgh Univ., October 1991. Also in *Logic and Algebra of Specification*, ed. F.L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag, 1993.
23. R. Milner. Action structure for the π-calculus. Technical Report ECS–LFCS–93–264, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1992.
24. R. Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.
25. R.. Milner. Action calculi, or syntactic action structures. In *Proc MFCS'93*, volume 711 of *Lecture Notes in Computer Science*, pages 105–121. Springer Verlag, 1993.
26. R. Milner. Action calculi V: reflexive molecular forms (with appendix by Ole Jensen). Draft, 1994.
27. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.

28. R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *19th ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer Verlag, 1992.
29. U. Nestmann and B. Pierce. Decoding choice encodings. In *Proc. CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.
30. G.K. Ostheimer and A.J.T. Davie. π-calculus characterisations of some practical λ-calculus reductions strategies. Technical Report CS/93/14, St. Andrews, 1993.
31. J. Parrow and B. Victor. The update calculus. In *Proc. AMAST '97*, volume 1349 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.
32. J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. To appear in *Proc. LICS'98*, IEEE Computer Society Press., 1998.
33. B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. Technical Report CSCI 476, Indiana University, 1997. To appear in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, MIT Press.
34. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh, 1992.
35. D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Information and Computation*, 111(1):120–153, 1994.
36. D. Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. *Theoretical Computer Science*, 155:39–83, 1996.
37. D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(2):235–274, 1996.
38. D. Sangiorgi. The name discipline of receptiveness. In *24th ICALP*, volume 1256 of *Lecture Notes in Computer Science*. Springer Verlag, 1997. Full paper available electronically as `ftp://zenon.inria.fr/meije/theorie-par/davides/Receptiveness.ps.Z`.
39. H. Thielecke. *Categorical Structure of Continuation Passing Style.* PhD thesis, Department of Computer Science, University of Edinburgh, 1997.
40. F. van Breugel. A Labelled Transition System for the $\pi\epsilon$-calculus. In *Proc. of the 7th International Joint Conference on the Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 312–336. Springer Verlag, April 1997.
41. N. Yoshida. Minimality and Separation Results on Asynchronous Mobile Processes: representability theorem by concurrent combinators. Submitted for publication, 1998.
42. Nobuko Yoshida. Graph types for monadic mobile processes. In *Proc. FST & TCS*, volume 1180 of *Lecture Notes in Computer Science*, pages 371–386. Springer Verlag, 1996.