On asynchrony in name-passing calculi*

Massimo Merro
COGS, University of Sussex, UK

Davide Sangiorgi
INRIA, Sophia-Antipolis, France

October 26, 2001

Abstract

The asynchronous π-calculus is considered the basis of experimental programming languages (or

guarded choice [38]), it seems fair to say that the restriction to asynchronous contexts does not allows us to gain much.

By contrast, asynchrony has strong semantic consequences under simplifications a) and b). Consider the following laws which are valid under the specified conditions) in Lπ, but are false in π_a and in π-calculus:

$$\overline{a}b \;=\; \boldsymbol{\nu}c)\ \overline{a}c \mid !c\ x).\,\overline{b}x) \qquad 1)$$

$$\boldsymbol{\nu}a)\ !a\ x).\,R \mid P \mid Q) \;=\; \boldsymbol{\nu}a)\ !a\ x).\,R \mid P) \mid\ \boldsymbol{\nu}a)\ !a\ x).\,R \mid Q) \qquad 2)$$

$$\boldsymbol{\nu}a)\ !a\ x).\,R \mid C[\overline{a}b]) \;=\; \boldsymbol{\nu}a)\ !a\ x).\,R \mid C[R\{{}^{b}\!/\!_{x}\}]) \qquad 3)$$

$\boldsymbol{\nu}c)$ $\overline{}$

Proof techniques for Lπ can be exploited to reason about languages such as Pict, Join, and TyCO, either by directly adapting the techniques to these languages, or by means of encodings into Lπ. The theory of Lπ for instance, its algebraic properties and labelled bisimulations) is also useful in calculi where the usage of some names goes beyond the syntax of Lπ. For instance, there could be a distinct set of synchronous names, or names that can be tested for identity see, for instance, [30]). 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 formers.

For simplicity we develop the theory for a *monadic* calculus where exactly one name may be transmitted); the generalisation to the *polyadic* version where tuple of names may be transmitted) is straightforward.

1.1 Related work

Calculi similar to Localised π are discussed by Honda and Tokoro [22], Amadio [2], Boreale [7], and Yoshida [62]. A number of characterisations of barbed congruence on asynchronous mobile processes exist [3, 8, 2]. However, in the labelled bisimilarities used, matching transitions of processes have the same labels, therefore laws like 1–5) do not hold.

Other studies on barbed congruence, or similar context-based bisimulations, for mobile processes have been conducted, for instance by Honda and Yoshida [24, 61] Kobayashi, Pierce, and Turner [27], Hennessy and Riely [19] and, for a coordination language, by Busi, Gorrieri, and Zavattaro [12]). Boreale and Sangiorgi [9] have studied barbed congruence in synchronous π-calculus with capability types and no matching, where Lπ can be treated as 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] bisimilarities have a type environment in fact, closures) whereas our bisimilarities are directly defined on processes.

By the time the writing of this paper has been completed, the theory of Lπ has already been used in some works. In [29], the first author gives an encoding of polyadic Lπ into monadic Lπ. Unlike Milner's encoding of polyadic π into monadic π [31], the encoding in [29] is fully-abstract with respect to barbed congruence. In [54], the second author gives a fully-abstract encoding of *higher-order* Lπ where processes can be transmitted) into Lπ. The theory of Lπ allows proofs simpler than those of analogous results for other π-calculi [46, 49]. Finally, in [30] Lπ is used to give a translational semantics of an appropriate abstraction of) Cardelli's distributed object-based programming language *Obliq* [13]. The theory of Lπ precisely a typed variant of Lemma 5.17) is used to prove the correctness of *object migration*.

In Join and Blue calculus, polymorphic type systems à la ML have been introduced [17, 14]. In both cases, the constraint on the output capability of names is crucial. We believe that similar polymorphic type systems can be defined in Lπ.

1.2 Outline

In Section 2 we give syntax and operational semantics of Lπ. In Section 3 we recall some common bisimulation-based behavioural equivalences for π-calculi. In Section 4 we present some special processes, the *link processes*, which are important in the theory of Lπ. In Section 5 we give the first proof technique for barbed congruence. In Section 6 we give the second proof technique for barbed congruence. In Section 7 we prove that these two proof techniques completely describe barbed congruence. In Section 8 we enhance the second proof technique with a new form of up-to proof technique. Section 9 is entirely devoted to applications: in subsection 9.1 we use link processes to express name substitutions; in Subsection 9.3 we prove that the *delayed input* a form of non-blocking input prefixing) is derivable in Lπ, and present some of its algebraic properties. In Subsection 9.2 we prove a sharpened form of Milner's *replication theorems* [31]. In Subsection 9.4 we give an optimisation of the encoding of call-by-name λ-calculus and, exploiting delayed input, we derive an encoding of *strong call-by-name*. In Subsection 9.5 we prove some laws for Fournet and Gonthier's Join-calculus [15]. In Subsection 9.6 we prove some non-full abstraction

inp:

In π-calculi, barbed congruence coincides with the closure under substitutions of early bisimilarity [46, 3]. In asynchronous calculi *without* matching, like Lπ, early bisimilarity is a congruence and it coincides with its simpler *ground* variant [20, 48], which differ from the early one in that there is no universal quantification in the input clause.

Definition 3.3 *A symmetric relation $\mathcal{S}$ on π-terms is an* oτ-bisimulation *if $P\ \mathcal{S}\ Q$, $P \xrightarrow{\mu} P'$, μ is not an input and* bn $\mu) \cap$ fn $Q) = \emptyset$*, implies that there exists Q' such that $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ and $P'\ \mathcal{S}\ Q'$.*

Definition 3.4 (Ground bisimilarities)

- Synchronous ground bisimulation *is the largest oτ-bisimulation $\mathcal{S}$ on processes such that $P\ \mathcal{S}\ Q$ and $P \xrightarrow{a(b)} P'$, with $b \notin$ fn Q), implies that there exists Q' such that $Q \stackrel{a(b)}{=\!=}$*

(a) either $P \xrightarrow{a(b)} P'$ *and* $P' \mathcal{S} Q'$

(b) or $P \xrightarrow{\hat{\tau}} P'$ *and* $P \mid \overline{a}b) \mathcal{S} Q'$.

We write $P \lesssim_{\mathrm{a}} Q$, *if* $P \mathcal{S} Q$ *for some asynchronous ground expansion* $\mathcal{S}$.

4 The link processes

The theory of Lπ is based on special processes called *links* which behave as name buffers receiving names at one end-point and retransmitting them at the other end-point in the π-calculus literature, links are sometimes called forwarders [24] or wires [56]).

Definition 4.1 (Static link) *Given any two names* a *and* b, *we call* static link *the process below:*

$$a \triangleright b \stackrel{\text{def}}{=} !a\ x).\, \overline{b}x.$$

We sometimes use a more sophisticated form of link $a \to b$, 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 emit private names [50]).

Definition 4.2 (Dynamic link) *Given two names* a *and* b, *we call* dynamic link *the process defined by the following recursive definition:*

$$a \to b \stackrel{\text{def}}{=} !a\ x).\ \boldsymbol{\nu}c)\ \overline{b}c \mid c \to x).$$

Being recursively defined, the process $a \to b$ is not in Lπ. However, there exists a process in Lπ which is synchronous bisimilar to it. In the following we explain how this process can be built up. In [31], Milner shows that recursive definitions can be encoded, up to bisimilarity, in terms of replication. When applying Milner's encoding to a dynamic link we get a processes which does not respect the Lπ constraint on the output capability. This problem can be avoided by rewriting the definition of dynamic links thus:

$$a \to b \stackrel{\text{def}}{=} !a\ x).\, \text{out}\ b, x)$$

where out $b, x)$ is recursively defined as:

$$\text{out}\ b, x) \stackrel{\text{def}}{=} \boldsymbol{\nu}c)\ \overline{b}c \mid !c\ y$$

(a) *either* $Q \stackrel{a(b)}{\Longrightarrow} Q'$ *and* $P' \mathcal{S} Q'$

(b) *or* $Q \Longrightarrow Q'$ *and* $P' \mathcal{S} \quad Q' \mid [\![\overline{a}b]\!]$).

Processes P *and* Q *are* $\asymp_{\mathrm{a}}$*-bisimilar, written* $P \asymp_{\mathrm{a}} Q$*, if* $P \mathcal{S} Q$ *for some* $\asymp_{\mathrm{a}}$*-bisimulation* $\mathcal{S}$.

Relation $\asymp_{\mathrm{a}}$ is designed to be used on $\mathcal{L}\pi$-processes and therefore its definition does not contain clauses for free output actions. In order to prove that $\asymp_{\mathrm{a}}$ is a congruence relation over $\mathcal{L}\pi$ we adapt the up-to expansion proof technique of [55].

Definition 5.7 ($\asymp_{\mathrm{a}}$-bisimulation up to $\gtrsim$ and $\approx$) *A symmetric relation* $\mathcal{S}$ *is a* $\asymp_{\mathrm{a}}$-bisimulation up to $\gtrsim$ and $\approx$ *if whenever* $P \mathcal{S} Q$ *the following holds:*

1. *If* $P \stackrel{\tau}{\longrightarrow} P'$*, then there exists* Q' *such that* $Q \Longrightarrow Q'$ *and* $P' \gtrsim \mathcal{S} \approx Q'$.
2. *If* $P \stackrel{\overline{a}(b)}{\longrightarrow} P'$, $b \notin \mathrm{fn}\ Q$*), then there exists* Q' *such that* $Q \stackrel{\overline{a}(b)}{\Longrightarrow} Q'$ *and* $P' \gtrsim \mathcal{S} \approx Q'$.
3. *If* $P \stackrel{ab}{\longrightarrow} P'$, $b \notin \mathrm{fn}\ P, Q$*), then there exists* Q' *such that:*

   (a) *either* $Q \stackrel{ab}{\Longrightarrow} Q'$ *and* $P' \gtrsim \mathcal{S} \approx Q'$

   (b) *or* $Q \Longrightarrow Q'$ *and* $P' \gtrsim \mathcal{S} \approx \quad Q' \mid [\![\overline{a}b]\!]$).

Lemma 5.8 *If* $\mathcal{S}$ *is a* $\asymp_{\mathrm{a}}$*-bisimulation up to* $\gtrsim$ *and* $\approx$ *then* $\mathcal{S} \subseteq \asymp_{\mathrm{a}}$.

Proof: The proof is analogous to that in [55]. If $\mathcal{S}$ is a $\asymp_{\mathrm{a}}$-bisimulation up to $\gtrsim$ and $\approx$, then one shows that the relation $\approx \mathcal{S} \approx$ is a $\asymp_{\mathrm{a}}$-bisimulation. This follows from the transitivity of $\approx$ and the fact that $\approx$ is preserved by parallel composition The latter result is necessary to deal with clause 3b) of Definition 5.6). Finally, since $\mathcal{S} \subseteq \approx \mathcal{S} \approx \subseteq \asymp_{\mathrm{a}}$, we can conclude. □

In the sequel, we often use a $\asymp_{\mathrm{a}}$*-bisimulation up to* $\equiv$ proof technique. The soundness of this technique follows from Lemma 5.8 and the fact that $\equiv$ is contained in $\gtrsim$ and therefore also in $\approx$).

The following lemma gives us some information about the structure of the $\mathcal{L}\pi$-processes which may perform an output action. Part 2 of Lemma 5.9 will be needed to prove that $\asymp_{\mathrm{a}}$ is preserved by parallel composition.

Lemma 5. *Let* P *be an* $\mathcal{L}\pi$*-process.*

1. *If* $P \stackrel{\overline{a}(c)}{\longrightarrow} P_1$ *then* $P \equiv \ \boldsymbol{\nu}\widetilde{z})\ \ \boldsymbol{\nu}c)\ \overline{a}c \mid c \to b) \mid P_2)$ *and* $P_1 \equiv \ \boldsymbol{\nu}\widetilde{z})\ c \to b \mid P_2)$ *for some* $\widetilde{z}$, b *and* P_2 *such that* $\{a, c\} \cap \widetilde{z} = \emptyset$ *and* $c \notin \mathrm{fn}\ P_2$*).*
2. *If* $P \stackrel{\overline{a}(c)}{\longrightarrow} P_1$ *then* $P \approx \ \boldsymbol{\nu}c)\ [\![\overline{a}c]\!] \mid P_1)$.

Proof:

1. By transition induction.
2. By part 1 $P \equiv \ \boldsymbol{\nu}\widetilde{z})\ \ \boldsymbol{\nu}c)\ \overline{a}c \mid c \to b) \mid P_2)$ and $P_1 \equiv \ \boldsymbol{\nu}\widetilde{z})\ c \to b \mid P_2)$ for some $\widetilde{z}$, b and P_2 such that $\{a, c\} \cap \widetilde{z} = \emptyset$ and $c \notin \mathrm{fn}\ P_2$). Picking some fresh name d we have, using proposition 4.3 2):

$$\begin{aligned} P &\equiv \ \boldsymbol{\nu}\widetilde{z})\ \ \boldsymbol{\nu}d)\ \overline{a}d \mid d \to b) \mid P_2) \\ &\approx \ \boldsymbol{\nu}\widetilde{z})\ \ \boldsymbol{\nu}d)\ \overline{a}d \mid \ \boldsymbol{\nu}c)\ d \to c \mid c \to b)) \mid P_2) \\ &\equiv \ \boldsymbol{\nu}c)\ \ \boldsymbol{\nu}d)\ \overline{a}d \mid d \to c) \mid \ \boldsymbol{\nu}\widetilde{z})\ c \to b \mid P_2)) \\ &\equiv \ \boldsymbol{\nu}c)\ [\![\overline{a}c\ P \end{aligned}$$

Lemma 5.10 *Let P and Q be two $\mathcal{L}\pi$-processes such that $P \asymp_{\mathrm{a}} Q$. Then:*

1. $\boldsymbol{\nu}a)P \asymp_{\mathrm{a}} \boldsymbol{\nu}a)Q$
2. $P \mid R \asymp_{\mathrm{a}} Q \mid R$, *for all* $\not\leftrightarrow a$

2. We prove that the relation $\mathcal{S}$ defined below

$$\{\ P_1 \mid !a\ x).Q_1, P_2 \mid !a\ x).Q_2) : P_1, P_2$$

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

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

$$\texttt{sync:}\ \frac{P \xrightarrow{\tau} P'}{P \overset{\tau}{\longmapsto} P'}$$

$$\texttt{input:}\ P \overset{a(b)}{}$$

(a) *either* $Q \stackrel{\overline{a}d}{\Longrightarrow} Q'$ *and* $\exists\, p \notin \mathrm{fn}\ P, Q)$ *such that* $p \triangleright b \mid P') \ \mathcal{S}\ p \triangleright d \mid Q')$

(b) *or* $Q \stackrel{\overline{a}(c)}{}$

7 Two

$R\ n, \mathcal{L}) =$

8 Up-to link proof techniques

As for standard bisimilarity, link bisimilarity can be enhanced by means of up to expansion techniques [55].

Definition 8.1 (Link bisimilarity up to expansion) *A symmetric relation $\mathcal{S}$ on* Lπ*-processes is a* link bisimulation up to $\gtrsim$ *if whenever $P\ \mathcal{S}\ Q$ the following holds:*

1. *If $P \xrightarrow{\tau} P'$, then $Q \Longrightarrow Q'$ and $P' \gtrsim \mathcal{S} \lesssim Q'$.*
2. *If $P \xrightarrow{a(p)} P'$, and $p \notin \text{fn}\ Q)$, then*
   (a) *either $Q \overset{a(p)}{\Longrightarrow} Q'$ and $P' \gtrsim \mathcal{S} \lesssim Q'$*
   (b) *or $Q \Longrightarrow Q'$ and $P' \gtrsim \mathcal{S} \lesssim\ Q' \mid \overline{a}p)$.*
3. *If $P \xrightarrow{\overline{a}b} P'$, and $p \notin \text{fn}\ P, Q)$, then*
   (a) *either $Q \overset{\overline{a}d}{\Longrightarrow} Q'$ and $p \triangleright b \mid P') \gtrsim \mathcal{S} \lesssim\ p \triangleright d \mid Q')$*
   (b) *or $Q \overset{\overline{a}(c)}{\Longrightarrow} Q'$, with $c \notin \text{fn}\ P)$, and $p \triangleright b \mid P') \gtrsim \mathcal{S} \lesssim\ \boldsymbol{\nu} c)\ p \triangleright c \mid Q')$.*
4. *If $P \xrightarrow{\overline{a}(c)} P'$, with $c \notin \text{fn}\ Q)$ and $p \notin \text{fn}\ P, Q)$, then*
   (a) *either $Q \overset{\overline{a}b}{\Longrightarrow} Q'$ and $\boldsymbol{\nu} c)\ p \triangleright c \mid P') \gtrsim \mathcal{S} \lesssim\ p \triangleright b \mid Q')$*
   (b) *or $Q \overset{\overline{a}(c)}{\Longrightarrow} Q'$ and $\boldsymbol{\nu} c)\ p \triangleright c \mid P') \gtrsim \mathcal{S} \lesssim\ \boldsymbol{\nu} c)\ p \triangleright c \mid Q')$.*

Lemma 8.2 *If $\mathcal{S}$ is a link bisimilarity up to expansion then $\mathcal{S} \subseteq \approx_l$.*

Proof: The proof is analogous to that of standard bisimilarity [55]. □

Link bisimilarity can be strengthened by means of a more powerful up-to proof technique which allows us to reduce the number of links introduced in the derivatives. Roughly speaking, when comparing two processes P and Q this technique permits cutting the *same* links from both processes, or to cut a *private* link from one process only.

Definition 8.3 (Link bisimulation up to link) *A symmetric relation $\mathcal{S}$ on* Lπ*-processes is a* link bisimulation up to link *if whenever $P\ \mathcal{S}\ Q$ the following holds:*

1. *If $P \xrightarrow{\tau} P'$ then $Q \Longrightarrow Q'$ and $P'\ \mathcal{S}\ Q'$.*
2. *If $P \xrightarrow{a(p)} P'$, and $p \notin \text{fn}\ Q)$, then*
   (a) *either $Q \overset{a(p)}{\Longrightarrow} Q'$ and $P'\ \mathcal{S}\ Q'$*
   (b) *or $Q \Longrightarrow Q'$ and $P' \mathcal{S}\ Q' \mid \overline{a}p)$.*
3. *If $P \xrightarrow{\overline{a}b} P'$, and $p \notin \text{fn}\ P, Q)$, then*
   (a) *either $Q \overset{\overline{a}b}{\Longrightarrow} Q'$ and $P'\ \mathcal{S}\ Q'$*
   (b) *or $Q \overset{\overline{a}d}{\Longrightarrow} Q'$ Q and $p \triangleright b \mid P' \triangleright$ b*

4.

9 Applications of thethe

9.2 The replication theorems

The *replication theorems* [31] express useful distributivity properties of private replicated processes. The assertions of the theorems can be read thus: A passive resource that is shared among a certain number of clients can be made private to each of them.

Theorem .3 (Standard replication theorems) *Assume that name a occurs free in processes P, Q and R only in output subject position. Then:*

1. $\boldsymbol{\nu} a)\big(!a\ x).R \mid P \mid Q\big) \approx \boldsymbol{\nu} x)\ !a\ x).R \mid P) \mid \boldsymbol{\nu} a)\ !a\ x).R \mid Q)$.
2. $\boldsymbol{\nu} a)\ !a\ x).R \mid !P) \approx !\ \boldsymbol{\nu} a)\ !a\ x).R \mid P)$.

The side condition in the theorems prevents the restricted name a from being exported. As a consequence, the theorem cannot be used in situations where the set of clients of the resource $a\ x).R$ may change dynamically. To see why this side condition is necessary, take:

$$\begin{array}{lcl} P_1 & \stackrel{\text{def}}{=} & \boldsymbol{\nu} a)\ !a\ x).R \mid \overline{b}a \mid Q) \\ P_2 & \stackrel{\text{def}}{=} & \boldsymbol{\nu} a)\ !a\ x).R \mid \overline{b}a) \mid \boldsymbol{\nu} a)\ !a\ x).R \mid Q) \end{array}$$

These processes are in general not equivalent in π-calculus. Intuitively, the environment external to P_1 can receive a along b and then use it in input position to interfere with an attempt by Q to activate a copy of R. This is not possible in P_2, where Q has its own private access to R. The difference between P_1 and P_2 can be observed in a context that receives a and then uses it in input; in this way, the context may steal messages that were supposed to reach the resource.

Pierce and Sangiorgi [43] have shown that the side condition can be relaxed using the type system with input/output capabilities, and requiring that the processes R, P and Q only possess the output capability on a. The same result is proved in [52] by previously translating processes by means of an encoding very close to our $[\![\cdot]\!]$) and then proving that the image are bisimilar. In both cases the sharpened replication theorems are shown valid with respect to typed) barbed congruence by proving a few barbed bisimilarities. Here we propose easier proofs of the sharpened replication theorems without using typed bisimulations. While the results in [43, 52] are for the standard) π-calculus, our results only apply

$$\texttt{d-in:}\frac{}{a\ b)P \xrightarrow{a(b)} P} \qquad \texttt{s-com:}\frac{P \xrightarrow{\overline{a}c} P'}{a\ b)P \xrightarrow{\tau} \boldsymbol{\nu} b)\ P'\{c/b\})}$$

$$\texttt{p-in:}\frac{P \xrightarrow{\mu} P' \quad b \notin \mathrm{n}\ \mu) \quad a \notin \mathrm{bn}\ \mu)}{a\ b)P \xrightarrow{\mu} a\ b)P'} \qquad \texttt{s-cls:}\frac{P \xrightarrow{\beta_c \overline{a}c} P' \quad b \notin \mathrm{n}\ \beta_c \overline{a}c)}{a\ b)P \xrightarrow{\tau} \beta_c\ P'\{c/b\})}$$

$$\texttt{o-}\boldsymbol{\nu}\texttt{:}\frac{P \xrightarrow{\mu} P' \quad [\mu = \overline{a}c \vee \mu = c\ b)\overline{a}b] \quad a \neq c}{\boldsymbol{\nu} c)P \xrightarrow{(\boldsymbol{\nu} c)\mu} P'} \qquad \texttt{o-in:}\frac{P \xrightarrow{\overline{a}b} P' \qquad b \neq a}{c\ b)P \xrightarrow{c(b)\overline{a}b} P'}$$

$$\texttt{cls:}\ P \xrightarrow{\beta_c \overline{a}c} P' \quad Q \xrightarrow{a(b)} Q' \quad \mathrm{bn}\ \beta_c) \cap \mathrm{fn}\ Q) =$$

$$\{\!|\, P_1 \mid P_2 \,|\!\} \stackrel{\text{def}}{=} \{\!|\, P_1 \,|\!\} \mid \{\!|\, P_2 \,|\!\} \qquad \{\!|\, \boldsymbol{\nu} a) P \,|\!\} \stackrel{\text{def}}{=} \boldsymbol{\nu} a) \{\!|\, P \,|\!\} \qquad \{\!|\, \overline{a}b \,|\!\} \stackrel{\text{def}}{=} \overline{a}b$$

$$\{\!|\, a\ b). P \,|\!\} \stackrel{\text{def}}{=} a\ b). \{\!|\, P \,|\!\} \qquad \{\!|\, !a\ b). P \,|\!\} \stackrel{\text{def}}{=} !a\ b). \{\!|\, P \,|\!\} \qquad \{\!|\, \mathbf{0} \,|\!\} \stackrel{\text{def}}{=} \mathbf{0}$$

$$\{\!|\, a\ b) P \,|\!\} \stackrel{\text{def}}{=} \boldsymbol{\nu} b)\ a\ b'). b \triangleright b' \mid \{\!|\, P \,|\!\})$$

Table 5: The encoding $\{\!|\cdot|\!\}$

---

between processes P and $\{\!|\, P \,|\!\}$, up to some notion of expansion. As we will argue in Remark 9.11, such an operational correspondence is not easy to prove. Thus, for convenience, we define an auxiliary encoding $\{\!|[\cdot]|\!\}$ as the composition of the encodings $\{\!|\cdot|\!\}$ and $[\![\cdot]\!]$ of Section 5). Precisely, if P is a DLπ-process

$$\{\!|[P]|\!\} \stackrel{\text{def}}{=} [\![\{\!|\, P \,|\!\}]\!].$$

We prove that the encoding $\{\!|[\cdot]|\!\}$ satisfies an operational correspondence up to $\succcurlyeq_{\mathrm{a}}$, where $\preccurlyeq_{\mathrm{a}}$ denotes the expansion variant of $\asymp_{\mathrm{a}}$ Definition 5.6) in the same way as $\lesssim_{\mathrm{a}}$ denotes the expansion variant of $\approx_{\mathrm{a}}$ see Definition 3.5). This operational correspondence, together with Theorem 5.3, allows us to prove the soundness of $\{\!|\cdot|\!\}$. Then, by exploiting the inclusion L$\pi \subset$ DLπ we prove the completeness of $\{\!|\cdot|\!\}$.

For proving the operational correspondence of $\{\!|[\cdot]|\!\}$ we need to know that $\preccurlyeq_{\mathrm{a}}$ is a precongruence in $\mathcal{L}\pi$ simply adapt Corollary 5.13). We also need the following technical lemma.

Lemma .5 *Given an* Lπ*-process* P *and a name* a *it holds that*

$$\boldsymbol{\nu} a)\ a \to a \mid [\![P]\!]) \succcurlyeq_{\mathrm{a}} \boldsymbol{\nu} a)[\![P]\!].$$

Proof: We have $a \to a \succcurlyeq_{\mathrm{a}} \mathbf{0}$. Then we can conclude because $\succcurlyeq_{\mathrm{a}}$ is a precongruence. □

Lemma .6 (Operational correspondence of $\{\!|[\cdot]|\!\}$) *Let* P *be a process in* DLπ.

1. *Suppose that* $P \xrightarrow{\mu} P'$. *Then we have:*

   (a) *if* $\mu = a\ b)$ *then* $\{\!|[P]|\!\} \xrightarrow{a(b')} \gtrsim \{\!|[P']|\!\}\{b'/b\}$ *and* $b' \notin \mathrm{fn}\ P)$

   (b) *if* $\mu = \overline{a}b$ *then* $\{\!|[P]|\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}}\ c \to b \mid \{\!|[P']|\!\})$, *with* $c \notin \mathrm{fn}\ P)$

   (c) *if* $\mu = \boldsymbol{\nu} b)\overline{a}b$ *then* $\{\!|[P]|\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}} \boldsymbol{\nu} b)\ c \to b \mid \{\!|[P']|\!\})$, *with* $c \notin \mathrm{fn}\ P)$

   (d) *if* $\mu = d\ b)\overline{a}b$ *then* $\{\!|[P]|\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}} \boldsymbol{\nu} b)\ d\ b'). b \to b' \mid c \to b \mid \{\!|[P']|\!\})$, *with* $\{b', c\} \cap \mathrm{fn}\ P) = \emptyset$

   (e) *if* $\mu = \boldsymbol{\nu} d$

(b) if $\mu = \boldsymbol{\nu} c)\overline{a}c$ *then:*

i. either $P \xrightarrow{\overline{a}b} P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} \ c \to b \mid \{\!|P'|\!\})$, *with* $c \notin \mathrm{fn}\ P)$

ii. or $P \xrightarrow{(\boldsymbol{\nu} b)\overline{a}b} P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} \ \boldsymbol{\nu} b)\ c \to b \mid \{\!|P'|\!\})$, *with* $c \notin \mathrm{fn}\ P)$

iii. or $P \xrightarrow{d(b)\overline{a}b} P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} \ \boldsymbol{\nu} b)\ d\ b').\, b \to b' \mid c \to b \mid \{\!|P'|\!\})$, *with* $\{b', c\} \cap \mathrm{fn}\ P) = \emptyset$

iv. or $P \xrightarrow{(\boldsymbol{\nu} d)d(b)\overline{a}b} P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} \boldsymbol{\nu} b)\ \boldsymbol{\nu} d)\ d\ b').\, b \to b' \mid c \to b \mid \{\!|P'|\!\})$, *with* $\{b', c\} \cap \mathrm{fn}\ P) = \emptyset$

(c) if $\mu = \tau$ *then* $P \xrightarrow{\tau} P'$ *with* $P_1 \succcurlyeq_{\mathrm{a}} \{\!|P'|\!\}$.

Proof: By transition induction. The proof relies on Proposition 4.3 2), Lemma 5.2, and Lemma 9.5.

Remark .11 *In order to clarify the usefulness of the auxiliary encoding $\{\!|\cdot|\!\}$, notice that if one wanted to prove the operational correspondence of $\{\!|\cdot|\!\}$ (instead of $\{\!|\cdot|\!\}$), then Proposition 4.3(2) and Lemma 9.5 would not be necessary anymore. By contrast, an expansion variant of Proposition 9.1 would be necessary. The proof of a such a result would require an expansion variant of Theorem 5.14 (or Corollary 5.15), which, as already pointed out in Remark 5.16, does not hold.*

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

$$\begin{array}{rcll} a\ b).\ P \mid Q) & = & a\ b)P) \mid Q & \text{if } b \notin \text{fn } Q) \qquad 9) \\ a\ b)c\ d)P & = & c\ d)a\ b)P & \text{if } c \neq b \text{ and } d \neq a \qquad 10) \\ \boldsymbol{\nu}a).\ a\ x).\ \overline{a}x \mid P)) & = & \boldsymbol{\nu}x)P & \text{if } a \notin \text{fn } P) \qquad 11) \end{array}$$

Laws 9 and 10 are similar to structural rules for restriction. Similar laws have been proposed in [11]. Law 11 transforms a delayed input binder into a restriction binder it might be interesting to examine delayed input from within action calculi [34]; for instance, law 11 is reminiscent of the definition of restriction in reflexive action calculi [35]).

9.4 Encodings of the λ-calculus

In this example, we use polyadicity, which is straightforward to accommodate in the theory of Lπ. We write $\overline{a}\langle b_1 \ldots b_n\rangle$ for polyadic outputs and $a\ x_1, \ldots x_n).\ P$ for polyadic inputs. Below, we give ■ilner's encoding of call-by-name λ-calculus into π-calculus more precisely, the variant in [39]).

$$\begin{array}{rcl} |\ \lambda x.\ M\ |\!|_p & \stackrel{\text{def}}{=} & \boldsymbol{\nu}v).\ \overline{p}\langle v\rangle \mid v\ x, q).\ |\ M\ |\!|_q) \\ |\ x\ |\!|_p & \stackrel{\text{def}}{=} & \overline{x}\langle p\rangle \\ |\ MN\ |\!|_p & \stackrel{\text{def}}{=} & \boldsymbol{\nu}q)\Big(|\ M\ |\!|_q \mid q\ v).\ \boldsymbol{\nu}x).\ \overline{v}\langle x, p\rangle \mid !x\ r).\ |\ N\ |\!|_r)\Big) \end{array}$$

This is also an encoding into polyadic) Lπ. By applying Proposition 9.1, 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):

$$|\ My\ |\!|_p \stackrel{\text{def}}{=} \boldsymbol{\nu}q)\Big(|\ M\ |\!|_q \mid q\ v).\ \overline{v}\langle y, p\rangle\Big)$$

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

an

$$\langle\!\langle\, a\langle b\rangle \,\rangle\!\rangle \stackrel{\text{def}}{=} \overline{a}b \qquad \langle\!\langle\, P \mid Q \,\rangle\!\rangle \stackrel{\text{def}}{=} \langle\!\langle\, P \,\rangle\!\rangle \mid \langle\!\langle\, Q \,\rangle\!\rangle$$

$$\langle\!\langle\, \texttt{def}\ a\langle x\rangle \mid b\langle y\rangle = P_1\ \texttt{in}\ P_2 \,\rangle\!\rangle \stackrel{\text{def}}{=} \boldsymbol{\nu} ab)\ (!a\ x).\, b\ y).\, (\langle\!\langle\, P_1 \,\rangle\!\rangle \mid \langle\!\langle\, P_2 \,\rangle\!\rangle)$$

Table 6: ■apping of the Join calculus into Lπ

9.5 Some properties of the Join calculus

We apply the theory of Lπ to prove some laws in Fournet and Gonthier's Join calculus [15], a calculus for distributed and concurrent programming.

The Join calculus is an off-spring of the asynchronous π-calculus specifically designed to facilitate distributed implementations of channels mechanisms. The syntax of the core) Join calculus is given by the following grammar:

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

The particle $a\langle b\rangle$ denotes the asynchronous output of name b at channel a. $P_1 \mid P_2$ denotes two processes P_1 and P_2 running in parallel. The construct $\texttt{def}\ a\langle x\rangle \mid b\langle y\rangle = P_1\ \texttt{in}\ P_2$ is a sort of amalgamation of the operators of replication, parallel composition, and restriction, which allows to model the joint reception of values from different channels.

Free names and *bound names* of a process P, are

have to add a layer of "firewalls" to the encoding. We conjecture that the above encoding is fully-abstract with respect to barbed congruence as an encoding of Join into Lπ a similar conjecture is made by Fournet and Gonthier [15]). It is easy to prove soundness, and this suffices for using

In [7],

4. If $P \xrightarrow{\overline{a}(b)} P'$, by Lemma 5.1, P_1 exists such that $[\![P]\!] \ \overline{a}(c)$

DISTR) $L\{a \langle b$

4. *If* $M \rightsquigarrow^* N$ *then* $\lfloor M \rfloor \Longrightarrow\!\!\!\!\equiv \lfloor N \rfloor$.

5. *If*

A Proofs

A.1 Proofs of Lemmas 5.10 and 5.12

For the sake of clarity we restate the result as follows.

Lemma A.1 *Let P and Q be two $\mathcal{L}\pi$-processes such that $P \asymp_{\mathrm{a}} Q$. Then:*

1. $\boldsymbol{\nu} a)P \asymp_{\mathrm{a}} \boldsymbol{\nu} a)Q$
2. $P \mid R \asymp_{\mathrm{a}} Q \mid R$, *for all $\mathcal{L}\pi$-process R*
3. $a\ x).\, P \asymp_{\mathrm{a}} a\ x).\, Q$
4. $!a\ x).\, P \asymp_{\mathrm{a}} !a\ x).\, Q$.

Proof: 1. It suffices to show that the relation

$$\mathcal{S} = \{\ (\boldsymbol{\nu} a)P\,,\ \boldsymbol{\nu} a)Q) :\ P, Q \in \mathcal{L}\pi \text{ and } P \asymp_{\mathrm{a}} Q\}$$

is a $\asymp_{\mathrm{a}}$-bisimulation up-to structural congruence. The proof is easy because the output actions performed by an $\mathcal{L}\pi$-process are always bound. We work up to structural congruence when dealing with the asynchronous clause for input.

2. We prove that the relation

$$\mathcal{S} = \{\ \boldsymbol{\nu}\tilde{a})\ P \mid R),\ \boldsymbol{\nu}\tilde{a})\ Q \mid R)) \text{ 10.12Tf6.480080Td a)Tj/R2100.12Tf5.280080P}$$

(d) if $\alpha = \tau$ *then* $[\![P]\!] \stackrel{\tau}{\Longrightarrow} \gtrsim [\![P']\!]$.

2. *Suppose that* $[\![P]\!] \stackrel{\alpha}{\Longrightarrow} P_1$. *Then there exists* $P' \in \mathrm{L}\pi$ *such that:*

   (a) if $\alpha = a\ b)$ *then* $P \stackrel{a(b)}{\Longrightarrow} P'$, *with* $P_1 \gtrsim [\![P']\!]$;

i. There exists Q' such that $Q \stackrel{\overline{a}b}{\Longrightarrow} Q'$ and $(\boldsymbol{\nu} c)\ p \triangleright c \mid P') \approx_{\mathrm{l}} (\boldsymbol{\nu} c)\ p \triangleright b \mid Q')$. By Lemma A.2, there exists a process Q_1 such that $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim (\boldsymbol{\nu} c)\ p \to b \mid Q')$. So, $[\![P]\!] \stackrel{\overline{a}(p)}{\longrightarrow} P_1 \gtrsim [\![(\boldsymbol{\nu} c)\ p \triangleright c \mid P')]\!]$ and $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim [\![p \triangleright b \mid Q']\!]$, and we can conclude since $([\![(\boldsymbol{\nu} c)\ p \triangleright c \mid P')]\!], [\![p \triangleright b \mid Q']\!]) \in \mathcal{S}$.

ii. There exists Q' such that $Q \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$ and $(\boldsymbol{\nu} c)\ p \triangleright c \mid P') \approx_{\mathrm{l}} (\boldsymbol{\nu} c)\ p \triangleright c \mid Q')$. By Lemma A.2, there exists a process Q_1 such that $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim (\boldsymbol{\nu} c)\ p \to c \mid [\![Q']\!])$. So, $[\![P]\!] \stackrel{\overline{a}(p)}{\longrightarrow} P_1 \gtrsim [\![(\boldsymbol{\nu} c)\ p \triangleright c \mid P')]\!]$ and $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim [\![(\boldsymbol{\nu} c)\ p \triangleright c \mid Q')]\!]$, and we can conclude since $([\![(\boldsymbol{\nu} c)\ p \triangleright c \mid P')]\!], [\![(\boldsymbol{\nu} c)\ p \triangleright c \mid Q')]\!]) \in \mathcal{S}$.

We prove the implicationj

b) There exists Q' such that $Q \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$ and $Q_1 \gtrsim \boldsymbol{\nu} c)\ p \to c \mid [\![Q']\!])$. So, we have $[\![\boldsymbol{\nu} c)\ p \triangleright c \mid P')]\!] \lesssim P_1 \asymp_{\mathrm{a}} Q_1 \gtrsim [\![\boldsymbol{\nu} c)\ p \triangleright c \mid Q')]\!]$. We can therefore conclude that $\boldsymbol{\nu} c)\ p \triangleright c \mid P'),\quad \boldsymbol{\nu} c)\ p \triangleright c \mid Q')) \in \mathcal{S}$.

□

A.3 Complement to the Proof of Lemma 7.1

Proof: The proof is by induction on n. The case $n = 0$ is trivial because $\simeq_{\mathrm{l}}^{0} = \mathrm{L}\pi \times \mathrm{L}\pi$. If $n > 0$, by induction, we suppose that

$$\boldsymbol{\nu}\mathcal{L}')\ P \mid R\ n, L)) \stackrel{\cdot}{\simeq} \boldsymbol{\nu}\mathcal{L}')\ Q \mid R\ n, L)) \text{ and } P \stackrel{\mu}{\Longrightarrow} P'.$$

We proceed by case analysis on the action μ to show that Q can match the action μ.

1. $\mu = \tau$. Then:

$$\boldsymbol{\nu}\mathcal{L}')\ P \mid R\ n, \mathcal{L}, \mathcal{M})) \stackrel{\tau}{\Longrightarrow} \boldsymbol{\nu}\mathcal{L}')\ P \mid\ \overline{c_n^{\tau}} \oplus R\ n-1, \mathcal{L}, \mathcal{M})))$$

To match this reduction up to barbed bisimulation we have to have:

$$\boldsymbol{\nu}\mathcal{L}')\ Q \mid R\ n, \mathcal{L}, \mathcal{M})) \stackrel{\tau}{\Longrightarrow} \boldsymbol{\nu}\mathcal{L}')\ Q_1 \mid\ \overline{c_n^{\tau}} \oplus R\ n-1, \mathcal{L}, \mathcal{M})))$$

We make a further reduction on the left handside

$$\boldsymbol{\nu}\mathcal{L}')\ P \mid\ \overline{c_n^{\tau}} \oplus R\ n-1, \mathcal{L}, \mathcal{M}))) \stackrel{\tau}{\Longrightarrow} \boldsymbol{\nu}\mathcal{L}')\ P' \mid R\ n-1, \mathcal{L}, \mathcal{M}))$$

Again this has to be matched by note that we cannot run the process $R\ n-1, \mathcal{L}, \mathcal{M})$ without losing a commitment $\overline{b_n}$ or $\overline{b'_n}$):

$$\boldsymbol{\nu}\mathcal{L}')\ Q_1 \mid\ \overline{c_n^{\tau}} \oplus R\ n-1, \mathcal{L}, \mathcal{M}))) \stackrel{\tau}{\Longrightarrow} \boldsymbol{\nu}\mathcal{L}')\ Q' \mid R\ n-1, \mathcal{L}, \mathcal{M}))$$

We observe that $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\tau}{\Longrightarrow} Q'$ and we conclude by applying the inductive hypothesis.

2. $\mu = \overline{a}b$. We may suppose $b \in L$. Then

$$\boldsymbol{\nu}\mathcal{L}')\ Q \mid R\ n, \mathcal{L}, \mathcal{M})) \quad \stackrel{\tau}{\Longrightarrow}$$
$$\boldsymbol{\nu}\mathcal{L}')\ Q_1 \mid \overline{c_n^{a}} \oplus a\ x).\ a' \triangleright x \mid R\ n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\})))$$

We take a further step on the lhs:

$$\boldsymbol{\nu}\mathcal{L}')\ P \mid \overline{c_n^{a}} \oplus a\ x).\ a' \triangleright x \mid R\ n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\},))) \quad \stackrel{\tau}{\Longrightarrow}$$
$$\boldsymbol{\nu}\mathcal{L}')\ P' \mid a' \triangleright b \mid R\ n-1, \mathcal{L}\ \mathcal{L}\ \mathcal{R}\ n$$

b) or by

$$\begin{aligned}\boldsymbol{\nu}\mathcal{L}')\ Q_1 \mid \overline{c_n^a} \oplus a\ x).\ a' \triangleright x \mid R\ n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\}))) \quad &\stackrel{\tau}{\Longrightarrow} \\ \boldsymbol{\nu}\mathcal{L}')\ \ \boldsymbol{\nu}c)\ Q' \mid a' \triangleright c) \mid R\ n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\}))\end{aligned}$$

This means that that $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$. By inductive

We make a further reduction on the lhs:

$$\boldsymbol{\nu}\mathcal{L}')\ P \mid\ \overline{\overline{c_n^a}} \oplus\ \ \boldsymbol{\nu}a')\ \overline{a}a' \mid R\ n-1, \mathcal{L}\cup\{a', \mathcal{M}\}))))) \quad \stackrel{\tau}{\Longrightarrow}$$
$$\boldsymbol{\nu}\mathcal{L}'a')\ P' \mid R\ n-1, \mathcal{L}\cup\{a', \mathcal{M}\}))$$

This is matched by:

$$\boldsymbol{\nu}\mathcal{L}')\ Q_1 \mid\ \overline{\overline{c_n^a}} \oplus\ \ \boldsymbol{\nu}a')\ \overline{a}a' \mid R\ n-1, \mathcal{L}\cup\{a', \mathcal{M}\}))))) \stackrel{\tau}{\Longrightarrow} Q''$$

We have two possibilities:

a) $Q_1 \stackrel{\tau}{\Longrightarrow} Q'$ and $Q'' \equiv\ \boldsymbol{\nu}\mathcal{L}'a')\ Q' \mid \overline{a}a' \mid R\ n-1, \mathcal{L}\cup\{a'\}, \mathcal{M}))$.

Then $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\tau}{\Longrightarrow} Q'$ and $P' \simeq_{\mathrm{l}}^{n-1} Q' \mid$

and

$$\begin{array}{rl} w \triangleright b \mid A_1) & \equiv \\ \boldsymbol{\nu} c)\ p \triangleright c \mid\ \ w \triangleright b \mid P')) & \mathcal{S} \\ \boldsymbol{\nu} c)\ p \triangleright c \mid\ \ \boldsymbol{\nu} d)\ w \triangleright d \mid Q'))) & \equiv \\ \boldsymbol{\nu} d)\ w \triangleright d \mid B_1). & \end{array}$$

iii. $Q \stackrel{\overline{a}h}{\Longrightarrow} Q'$, with $h \neq b$ and $w \triangleright b \mid P')\ \mathcal{S}\ \ w \triangleright h \mid Q')$ for some fresh name w. We reason as in the previous case.

d) If $A \xrightarrow{\overline{a}(b)} A_1$ the reasoning is similar to that for case 3.

3. The proof follows from Lemma A.6 2) and Proposition 4.3. Let r be a fresh name, by hypothesis we know that $P\{^r/_c\} \approx_{\text{lut}} \boldsymbol{\nu} c)\ r \triangleright c \mid Q)$. By Lemma A.6 2), we have

$$\boldsymbol{\nu} r)\ p \triangleright r \mid P\{^r/_c\}) \approx_{\text{lut}} \boldsymbol{\nu} r)\ p \triangleright r \mid\ \boldsymbol{\nu} c)\ r \triangleright c \mid Q))$$

for p fresh. By Proposition 4.3 we have

$$\boldsymbol{\nu} r)\ p \triangleright r \mid\ \boldsymbol{\nu} c)\ r \triangleright c \mid Q)) \equiv\ \boldsymbol{\nu} c)\ \ \boldsymbol{\nu} r)\ p \triangleright r \mid r \triangleright c) \mid Q) \gtrsim\ \boldsymbol{\nu} c)\ p \triangleright c \mid Q).$$

Since $\boldsymbol{\nu} c)\ p \triangleright c \mid P) \equiv\ \boldsymbol{\nu} r)\ p \triangleright r \mid P\{^r/_c\})$ and $\equiv \approx_{\text{lut}} \equiv \gtrsim \subset \approx_{\text{lut}}$ we can conclude that $\boldsymbol{\nu} c)\ p \triangleright c \mid P) \approx_{\text{lut}} \boldsymbol{\nu} c)\ p \triangleright c \mid Q)$.

4. As in Part 3, the proof can be derived by Lemma A.6 2) and Proposition 4.3.

□

A.5 Proof of Lemma 9.6

We restate Lemma 9.6.

Lemma A.7 (Operational correspondence of $\{\!|\cdot|\!\}$) *Let P be a process in* DLπ.

1. *Suppose that $P \xrightarrow{\mu} P'$. Then we have:*

   (a) *if $\mu = a\ b)$ then $\{\!|P|\!\} \xrightarrow{a(b')} \gtrsim \{\!|P'|\!\}\{^{b'}/_b\}$ and $b' \notin \text{fn}\ P)$*

   (b) *if $\mu = \overline{a}b$ then $\{\!|P|\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\text{a}}\ \ c \to b \mid \{\!|P'|\!\})$, with $c \notin \text{fn}\ P)$*

   (c) *if $\mu = \ \boldsymbol{\nu} b)\overline{a}b$ then $\{\!|P|\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\text{a}}\ \ \boldsymbol{\nu} b)\ c \to b \mid \{\!|P'|\!\})$, with $c \notin \text{fn}\ P)$*

   (d) *if $\mu = d\ b)\overline{a}b$ then $\{\!|P|\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\text{a}}\ \boldsymbol{\nu} b)\ d\ b').b \to b' \mid c \to b \mid \{\!|P'|\!\})$, with $\{b', c\} \cap \text{fn}\ P) = \emptyset$*

   (e) *if $\mu = \ \boldsymbol{\nu} d)d\ b)\overline{a}b$ then $\{\!|P|\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\text{a}}\ \ \boldsymbol{\nu} b)\ \boldsymbol{\nu} d)\ d\ b').b \to b' \mid c \to b \mid \{\!|P'|\!\})$, with $\{b', c\} \cap \text{fn}\ P) = \emptyset$*

   (f) *if $\mu = \tau$ then $\{\!|P|\!\} \xrightarrow{\tau} \succcurlyeq_{\text{a}} \{\!|P'|\!\}$.*

2. *Suppose that $\{\!|P|\!\} \xrightarrow{\mu} P_1$. Then there exists $P' \in$ DLπ such that:*

   (a) *if $\mu = a\ b')$ then $P \xrightarrow{a(b)} P'$ and $P_1 \succcurlyeq_{\text{a}} \{\!|P'|\!\}\{^{b'}/_b\}$*

   (b) *if $\mu = \ \boldsymbol{\nu} c)\overline{a}c$ then:*

      i. *either $P \xrightarrow{\overline{a}b} P'$ and $P_1 \succcurlyeq_{\text{a}}\ \ c \to b \mid \{\!|P'|\!\})$, with $c \notin \text{fn}\ P)$*

      ii. *or $P \xrightarrow{(\boldsymbol{\nu} b)\overline{a}b} P'$ and $P_1 \succcurlyeq_{\text{a}}\ \ \boldsymbol{\nu} b)\ c \to b \mid \{\!|P'|\!\})$, with $c \notin \text{fn}\ P)$*

iii. *or* $P \xrightarrow{d(b)\overline{a}b} P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} \boldsymbol{\nu} b)\ d\ b').\, b \to b' \mid c \to b \mid \{\!|P'|\!\})$,
with $\{b', c\} \cap \mathrm{fn}\ P) = \emptyset$

iv. *or* $P \xrightarrow{(\boldsymbol{\nu} d) d(b)\overline{a}b} P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} \boldsymbol{\nu} b)\ \boldsymbol{\nu} d)\ d\ b').\, b \to b' \mid c \to b \mid \{\!|P'|\!\})$,
with $\{b', c\} \cap \mathrm{fn}\ P) = \emptyset$

(c) *if* $\mu = \tau$ *then* $P \xrightarrow{\tau} P'$ *with* $P_1 \succcurlyeq_{\mathrm{a}} \{\!|P'|\!\}$.

Proof: The proof is by transition induction. We prove Part 1. The proof of Part 2 is similar.

1. $\mu = a\ b)$. The interesting case is when the last rule applied to get $P \xrightarrow{a(b)} P'$ is `d-in`. By Lemma 5.2 1), since $\gtrsim$ implies $\succcurlyeq_{\mathrm{a}}$, it holds that−∀∃ℛ∈∉{∈⇐∩⇒|ℛ∈∈{∈∀∀⇐⇐⇒|ℛ∈∉{∀∃∀⇐⇒|ℛ∈∈{∃∃∈

i. If $c \neq b$ then $\boldsymbol{\nu} b)\ P'\{c/b\}) = P'\{c/b\}$. By induction hypothesis it holds that

$$\{\!|\, P \,|\!\} \xrightarrow{(\boldsymbol{\nu} d)\overline{a}d} \succcurlyeq_{\mathrm{a}} d \to c \mid \{\!|\, P' \,|\!\}$$

with d fresh. Since $\{\!|\, a\ b)P \,|\!\} \stackrel{\text{def}}{=} \boldsymbol{\nu} b)\ a\ b').\, b \to b' \mid \{\!|\, P \,|\!\})$, by Proposition 4.3 2) and Lemma 5.2 it holds that:

$$\begin{aligned}\{\!|\, a\ b)P \,|\!\} \xrightarrow{\tau} &\succcurlyeq_{\mathrm{a}} \boldsymbol{\nu} b)\ \ \boldsymbol{\nu} d)\ b \to d \mid d \to c) \mid \{\!|\, P' \,|\!\})\\ &\succcurlyeq_{\mathrm{a}} \boldsymbol{\nu} b)\ b \to c \mid \{\!|\, P' \,|\!\})\\ &\succcurlyeq_{\mathrm{a}} \{\!|\, P' \,|\!\}\{c/b\}\\ &= \{\!|\, P\{c/b\} \,|\!\}.\end{aligned}$$

ii. If $c = b$ then $\boldsymbol{\nu} b)\ P'\{c/b\}) = \boldsymbol{\nu} b)P'$. By induction hypothesis it holds that

$$\{\!|\, P \,|\!\} \xrightarrow{(\boldsymbol{\nu} d)\overline{a}d} \succcurlyeq_{\mathrm{a}} d \to c \mid \{\!|\, P' \,|\!\}$$

with d fresh. Since $\{\!|\, a\ b)P \,|\!\} \stackrel{\text{def}}{=} \boldsymbol{\nu} b)\ a\ b').\, b \to b' \mid \{\!|\, P \,|\!\})$, by Proposition 4.3 2) and Lemma 9.5 it holds that:

$$\begin{aligned}\{\!|\, a\ b)P \,|\!\} \xrightarrow{\tau} &\succcurlyeq_{\mathrm{a}} \boldsymbol{\nu} b)\ \ \boldsymbol{\nu} d)\ b \to d \mid d \to b) \mid \{\!|\, P' \,|\!\})\\ &\succcurlyeq_{\mathrm{a}} \boldsymbol{\nu} b)\ b \to b \mid \{\!|\, P' \,|\!\})\\ &\succcurlyeq_{\mathrm{a}} \boldsymbol{\nu} b)\{\!|\, P' \,|\!\}.\end{aligned}$$

c) Suppose `cls` is the last rule applied for deriving $P \xrightarrow{\tau} P'$.

$$\texttt{cls:}\ \frac{P \xrightarrow{\beta_b \overline{a} b} P' \quad Q \xrightarrow{a\,(b')} Q' \quad \text{bn}\ \beta_b) \cap \text{fn}\ Q) = \emptyset}{P \mid Q \xrightarrow{\tau} \beta_b\ \ P' \mid Q'\{b/b'\})}$$

We can suppose $\beta_b = d\ b)$ for some d. The case when $\beta_b = \boldsymbol{\nu} d) d\ b)$ is similar. By induction hypothesis it holds that:

- $\{\!|\, P \,|\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}} \boldsymbol{\nu} b)\ d\ b').\, b \to b' \mid c \to b \mid \{\!|\, P' \,|\!\})$, with $\{b', c\} \cap fn\ P) = \emptyset$. $\{\, b$

References

[1] S. Abramsky. Proofs as Processes. *Theoretical Computer Science*, 135(1):5–9, December 1994.

[2] 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.

[3] R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195:291–324, 1998. Extended abstract in *Proc. CONCUR '96*, LNCS 1119, Springer Verlag.

[4] A. Appel. *Compiling with Continuations*. Cambridge University Press, 1992.

[5] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29:737–760, 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. *Theoretical Computer Science*, 195:205–226, 1998.

[8] M. Boreale, C. Fournet, and C. Laneve. Bisimulations for the Join Calculus. In *Proc. IFIP Conference PROCOMET'98*, 1997.

[9] M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. In *13th LICS Conf.* IEEE Computer Society Press, 1998.

[10] G. Boudol. Asynchrony and the π-calculus. Technical Report RR-1702, INRIA-Sophia Antipolis, 1992.

[11] 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.

[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] Luca Cardelli. A language with distributed scope. *Computing Systems*, 8(1):27–59, 1995. Short version in *Proc. of POPL '95*.

[14] Silvano Dal-Zilio. Implicit polymorphic type system for the blue calculus. Technical Report RR-3244, Inria, Institut National de Recherche en Informatique et en Automatique, 1997.

[15] C. Fournet and G. Gonthier. The Reflexive Chemical Abstract Machine and the Join calculus. In *Proc. 23th POPL*. ACM Press, 1996.

[16] Cedric Fournet. *The Join calculus: a Calculus for Distributed Mobile Programming*. PhD thesis, École Polytechnique, 1999.

[17] Cédric Fournet, Cosimo Laneve, Luc Maranget, and Didier Rémy. Implict typing à la ML for the join-calculus. In *Proc. CONCUR 97*, volume 1243 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[18] Y. Fu. A proof theoretical approach to communication. In *24th ICALP*, volume 1256 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[19] M. Hennessy and J. Riely. A typed language for distributed

[25] Hans Hüttel and Josva Kleist. Objects as mobile processes. Technical report RR-96-38, BRICS - Basic Research in Computer Science, 1996.

[26] J. Kleist and D. Sangiorgi. Imperative objects and mobile processes. In *Proc. PROCOMET'98*. North-Holland, 1998.

[27] N. Kobayashi, B.C. Pierce, and D.N. Turner. Linearity and the pi-calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, 1993. Short version in Proc. POPL'96.

[28] Jean-Jacques Lévy. Some results on the join-calculus. In *Proc. TACS '97*, volume 1281 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[29] M Merro. Locality and polyadicity in asynchronous name-passing calculi. In *Proc. FOSSACS2000*, volume 1784 of *Lecture Notes in Computer Science*. Springer Verlag, 2000.

[30] M. Merro, J. Kleist, and U. Nestmann. Mobile Objects as Mobile Processes. *Accepted for publication in Journal of Information and Computation*, 2001. An extended abstract entitled *Local π-calculus at Work: Mobile Objects as Mobile Processes*, in *Proceedings of TCS'00*, LNCS 1872, August 2000.

[31] 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.

[32] 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.

[33] R. Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[34] R.. Milner. Action calculi, or syntactic action structures. In *Proc MFCS'93*, volume 711 of *Lecture Notes in Computer Science*. Springer Verlag, 1993.

[35] R. Milner. Action calculi V: reflexive molecular forms (with appendix by Ole Jensen). Draft, 1994.

[36] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.

[37] R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. ICALP'92*, volume 623 of *Lecture Notes in Computer Science*. Springer Verlag, 1992.

[38] U. Nestmann and B. Pierce. Decoding choice encodings. *Journal of Information & Computation*, 163:1–59, November 2000.

[39] 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.

[40] J. Parrow and B. Victor. The update calculus. In *Proc. AMAST '97*, volume 1349 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[41] J. Parrow and B. Victor. The fusion calculus: -381.12 [illegible] 141.4erSt.Jensen)499.u9.92

[49]