Locality and Polyadicity in Asynchronous Name-Passing Calculi

Massimo Merro*

INRIA Sophia-Antipolis, France

Abstract. We give a *divergence-free* encoding of polyadic *Local* π into its monadic variant. Local π is a sub-calculus of asynchronous π-calculus where the recipients of a channel are local to the process that has created the channel. We prove the encoding *fully-abstract* with respect to barbed congruence. This implies that in Local π (i) polyadicity does not add extra expressive power, and (ii) when studying the theory of polyadic Local π we can focus on the simpler monadic variant. Then, we show how the idea of our encoding can be adapted to name-passing calculi with *non-binding input prefix*, such as *Chi*, *Fusion* and πF *calculi*.

1 Introduction

Local π, in short Lπ, is a variant of the asynchronous π-calculus [11, 5] where the recipients of a channel are local to the process that has created the channel. More precisely, in a process $(\boldsymbol{\nu} a)\, P$ all possible inputs at a appear – and are syntactically visible – in P; no further inputs may be created, inside or outside P. The locality property of channels is achieved by imposing that only the output capability of names may be transmitted, i.e., the recipient of a name may only use it in output actions. Lπ is a very expressive fragment of asynchronous π-calculus, and its theory has been studied in [15]; similar calculi are discussed, or at least mentioned, in [12, 4, 1, 30]. Lπ borrows ideas from some experimental programming languages (or proposals of programming languages), most notably Pict [20], Join [8], and Blue [6], and can be regarded as a basis for them (the restriction on output capabilities is not explicit in Pict, but, as we understand from the Pict users, most Pict programs obey it). The locality property makes Lπ particularly suitable for giving the semantics to, and reasoning about, concurrent or distributed object-oriented languages [14]. For instance, the locality property can guarantee unique identity of objects – a fundamental feature of objects.

As for most name-passing calculi, the theoretical developments on Lπ have been conducted on a monadic calculus, that is, a calculus in which only single names can be transmitted. On the other hand, most applications in name-passing calculi use polyadic communications, i.e., communications involving tuples of names. So, an interesting issue is to investigate whether monadic and polyadic name-passing calculi have the same expressive power. In this paper we show that,

* Funded by the European Union, under the Marie Curie TMR programme.

J. Tiuryn (Ed.): FOSSACS 2000, LNCS 1784, pp. 238–251, 2000.

under the locality hypothesis on channels, monadic and polyadic π-calculi have the same expressive power. More precisely, we give an encoding $\langle\!\langle \cdot \rangle\!\rangle$ of polyadic Lπ into monadic Lπ, and we prove it *fully-abstract* with respect to *barbed congruence* [18]. Our encoding is *divergence-free*, that is, it does not introduce infinite internal computations. Furthermore, we show how the idea of our encoding can be easily adapted to name-passing calculi with *non-binding input prefix*, such as *Chi calculus* [9], *Fusion calculus* [19] and πF-*calculus* [10], and we propose a simple encoding of polyadicity for these calculi.

The first attempt of encoding polyadicity in name-passing calculi is by Robin Milner [16]. Milner gives a simple encoding of polyadic into monadic synchronous π-calculus. Milner's encoding is not fully-abstract. In order to recover the full abstraction Yoshida [29], and Quaglia and Walker [21], have introduced two different *type systems* for monadic processes which model the communication protocol underlying Milner's encoding. A different approach has been followed by Gonthier and Fournet in the *Join-calculus* [8], an "extended subset" of the asynchronous π-calculus. In [8], among other results, a direct, although complex, fully-abstract encoding of polyadic processes into monadic ones is proposed. All these approaches will be discussed at the end of the paper.

In this extended abstract proofs are just sketched; complete proofs can be found in [13].

Outline The paper is structured as follows. In Section 2 we describe the polyadic Lπ calculus giving some properties of it; in Section 3 we recall a few correctness criteria for encodings; in Section 4 we present the encoding of polyadic Lπ into monadic Lπ; in Section 5 we prove the full abstraction of the encoding; in Section 6 we investigate other possible encodings of polyadicity in Lπ; in Section 7 we show how the idea of our encoding can be adapted in name-passing calculi with non-binding input prefix; in Section 8 we conclude and discuss related works.

2 The Polyadic Lπ

Polyadic Lπ, in short L$\widetilde{\pi}$, is an asynchronous fragment of Milner's polyadic π-calculus [16]. We use small letters $a, b, c, \ldots, x, y$ for *names*; capital letters P, Q, R for *processes*; and $\widetilde{a}$ to denote a tuple of names $a_1, \ldots, a_n$. L$\widetilde{\pi}$ has operators of inaction, input prefix, asynchronous output, parallel composition, restriction and replicated input:

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

where in input processes $a(\widetilde{x}).\, P$ names in $\widetilde{x}$ are all distinct and may not occur free in P in input position. This syntactic constraint ensures that only the output capability of names may be transmitted.

We use σ for *substitutions*; $P\sigma$ is the result of applying σ to P, with the usual renaming convention to avoid captures; $\{\widetilde{b}/\widetilde{a}\}$ is the simultaneous substitution of names $\widetilde{a}$ with names $\widetilde{b}$. Parallel composition has the lowest precedence

among the operators, and $\prod_n P_n$ is an abbreviation for the process $P_1 \mid \ldots \mid P_n$. We write $(\boldsymbol{\nu}\widetilde{a})\,P$ for $(\boldsymbol{\nu}a_1)\ldots(\boldsymbol{\nu}a_n)\,P$ and $\overline{a}b$ for $\overline{a}\langle b\rangle$. The *labeled transition system* is the usual one (in the late style [17]). *Structural congruence*, written $\equiv$ and defined as usual (see [16]), allows us to ignore certain structural differences between processes. Transitions are of the form $P \xrightarrow{\mu} P'$, where *action* μ can be: τ (interaction); $a(\widetilde{b})$ (input); $(\boldsymbol{\nu}\widetilde{c})\,\overline{a}\langle\widetilde{b}\rangle$ (output) where $\widetilde{c} \subseteq \widetilde{b}$ and $\overline{a}(\widetilde{b})$ is an abbreviation for $(\boldsymbol{\nu}\widetilde{b})\,\overline{a}\langle\widetilde{b}\rangle$. In these actions, a is the *subject* and $\widetilde{b}$ the *object*. We write $\xrightarrow{\hat{\mu}}$ to mean $P\xrightarrow{\mu}Q$, if $\mu \neq \tau$, and either $P = Q$ or $P\xrightarrow{\tau}Q$, if $\mu = \tau$. Relation $\Longrightarrow$ is the reflexive and transitive closure of $\xrightarrow{\tau}$; moreover, $\overset{\mu}{\Longrightarrow}$ stands for $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$, and $\overset{\hat{\mu}}{\Longrightarrow}$ for $\overset{\mu}{\Longrightarrow}$ if $\mu \neq \tau$, and for $\Longrightarrow$ if $\mu = \tau$. Free and bound names (fn, bn) of actions and processes are defined as usual.

We assume Milner's *sorting system*, under which all processes are well-sorted [16]. Names are partitioned into a collection of *sorts*. A *sorting function* is defined which maps sorts onto sequences of sorts. If a sort $\mathtt{S}$ is mapped onto a sequence of sorts $\widetilde{\mathtt{T}}$ this means that channels in $\mathtt{S}$ can only carry tuples in $\widetilde{\mathtt{T}}$. A sorting system is necessary to prevent arity mismatching in communications, like in $\overline{a}\langle b, c\rangle \mid a(x).\,P$. Substitutions must map names onto names of the same sort.

The *behavioral equivalence* we are interested in is *barbed congruence* [18]. It is well-known that barbed congruence represents a uniform mechanism for defining a behavioral equivalence in any process calculus possessing (i) an *interaction relation* (the τ-steps in π-calculus), modeling the evolution of the system, and (ii) an *observability predicate* $\downarrow_a$ for each name a which indicates the possibility for a process of accepting a communication at a with the environment. $P \downarrow_a$ holds if there is a derivative P', and an action μ, with subject a, such that $P\xrightarrow{\mu}P'$. We also write $P \Downarrow_a$ if there is a derivative P' such that $P \Longrightarrow P' \downarrow_a$. We recall that a *context* $C[\cdot]$ is a process with exactly one hole, written $[\cdot]$, where a process may be plugged in.

Definition 1 (barbed bisimilarity, congruence). Barbed bisimilarity, *written* $\dot{\approx}$, *is the largest symmetric relation on* π*-calculus processes such that* $P \dot{\approx} Q$ *implies:*

1. *If* $P\xrightarrow{\tau}P'$ *then there exists* Q' *such that* $Q \Longrightarrow Q'$ *and* $P' \dot{\approx} Q'$.
2. *If* $P \downarrow_a$ *then* $Q \Downarrow_a$.

Let $\mathcal{L}$ *be a set of processes in* π_{a}, *and* $P, Q \in \mathcal{L}$. *Two processes* P *and* Q *are* barbed congruent *in* $\mathcal{L}$, *written* $P \cong_{\mathcal{L}} Q$, *if for each context* $C[\cdot]$ *in* $\mathcal{L}$ *it holds that* $C[P] \dot{\approx} C[Q]$.

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. Simpler proof techniques are based on *labeled characterizations* without context quantification.

Definition 2 (ground bisimilarity). Ground bisimilarity, *written* $\approx$, *is the largest symmetric relation on processes such that if* $P \approx Q$, $P\xrightarrow{\mu}P'$, $\mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$, *then there exists* Q' *such that* $Q \overset{\hat{\mu}}{\Longrightarrow} Q'$ *and* $P' \approx Q'$.

We recall that in the asynchronous calculi without matching, like L$\widetilde{\pi}$, ground bisimilarity coincides with early, late, and open bisimilarities [23]. All these relations are congruences and imply barbed congruence.

In the technical part of this paper we shall need a means to count the number of silent moves performed by a process in order to use *up-to techniques* [27, 24]. The *expansion* relation [3], written $\lesssim$, is an asymmetric variant of $\approx$ such that $P \lesssim Q$ holds if $P \approx Q$, and Q has at least as many τ-moves as P.

Definition 3 (expansion). *$\lesssim$ is the largest relation on processes such that $P \lesssim Q$ implies:*

1. *whenever $P \xrightarrow{\mu} P'$, and $\mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$, there exists Q' such that $Q \stackrel{\mu}{\Longrightarrow} Q'$ and $P' \lesssim Q'$;*
2. *whenever $Q \xrightarrow{\mu} Q'$, and $\mathrm{bn}(\mu) \cap \mathrm{fn}(P) = \emptyset$, there exists P' such that $P \xrightarrow{\hat{\mu}} P'$ and $P' \lesssim Q'$.*

In both monadic and polyadic Lπ, barbed congruence is a relation strictly larger than ground bisimilarity. For instance, in Lπ, if $P = \overline{a}b$ and $Q = (\boldsymbol{\nu} c)\,(\overline{a}c \mid !c(x).\,\overline{b}x)$ then $P \cong_{\mathrm{L}\pi} Q$ (see [15]) but $P \not\approx Q$. In [15], Merro and Sangiorgi give two labeled characterizations of barbed congruence for monadic Lπ. One of them is based on an encoding of Lπ into πI, a calculus where all names emitted are private [25]. The (polyadic version of the) encoding (essentially Boreale's [4]) is an homomorphism on all operators except output, for which we have:

$$[\![\overline{a}\langle\widetilde{b}\rangle]\!] \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu}\widetilde{c})\,(\overline{a}\langle\widetilde{c}\rangle \mid \widetilde{c} \rightarrow \widetilde{b})$$

where $\widetilde{b} = (b_1, \ldots, b_n)$, $\widetilde{c} = (c_1, \ldots, c_n)$; names b_i and c_i have the same sort for all i; $\widetilde{c} \cap (\{a\} \cup \widetilde{b}) = \emptyset$; $\widetilde{c} \rightarrow \widetilde{b} \stackrel{\mathrm{def}}{=} \prod_{j=1}^{n} !c_j(\widetilde{x}).\,[\![\overline{b_j}\langle\widetilde{x}\rangle]\!]$ with $\widetilde{x} = (x_1, \ldots, x_{m_j})$.

Remark 1. Being recursively defined, the process $\widetilde{c} \rightarrow \widetilde{b}$ is not in L$\widetilde{\pi}$, but it is ground bisimilar to a process of L$\widetilde{\pi}$ (using replication instead of recursion).

Given two tuples of names $\widetilde{b} = (b_1, \ldots, b_n)$ and $\widetilde{c} = (c_1, \ldots, c_n)$ where names b_i and c_i have the same sort for all i, we denote with $\widetilde{c} \triangleright \widetilde{b}$ the process $\prod_{j=1}^{n} !c_j(\widetilde{x}).\,\overline{b_j}\langle\widetilde{x}\rangle$. Note that $[\![\widetilde{c} \triangleright \widetilde{b}]\!] = \widetilde{c} \rightarrow \widetilde{b}$.

Below, we report a simple adaption to the polyadic case of a few results on $[\![\cdot]\!]$ that have already appeared in the literature: Theorem 1 provides an adequacy result w.r.t. barbed bisimilarity; Theorem 2 gives a characterization of barbed congruence in L$\widetilde{\pi}$ for image-finite processes. We recall that the class of *image-finite processes* (to which most of the processes one would like to write belong) is the largest subset $\mathcal{I}$ of π-calculus process which is derivation closed and such that $P \in \mathcal{I}$ implies that, for all μ, the set $\{P' \;:\; P \stackrel{\mu}{\Longrightarrow} P'\}$, quotiented by alpha conversion, is finite.

Theorem 1 (Boreale [4]). *Let P and Q be two L$\widetilde{\pi}$-processes then*

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

Theorem 2 (Merro and Sangiorgi [15]). *Let P and Q be two $\mathrm{L}\widetilde{\pi}$-processes. Then*

1. $P \cong_{\mathrm{L}\widetilde{\pi}} Q$ implies $[\![P]\!] \approx [\![Q]\!]$, *for P and Q image-finite processes;*
2. $[\![P]\!] \approx [\![Q]\!]$ implies $P \cong_{\mathrm{L}\widetilde{\pi}} Q$.

Remark 2. Theorem 2 has been proved in [15] with respect to asynchronous barbed congruence (where only output barbs are taken into account) and an asynchronous variant of $\approx$. The adaptation to the synchronous case is straightforward.

3 Correctness Criteria for Encodings

When studying an encoding between two languages it is necessary to have some *correctness criteria* in order to assess the encoding. The most common correctness criteria for an encoding between two process calculi are based on the notions of *operational correspondence* and *full abstraction*. The former relates the execution steps as defined by an operational semantics of the source and target calculi. The latter relates the source and the target calculi at the level of behavioral equivalences. More formally, let us denote with $(\mathcal{S}, \asymp_{\mathrm{s}}, \longrightarrow_{\mathrm{s}})$ and $(\mathcal{T}, \asymp_{\mathrm{t}}, \longrightarrow_{\mathrm{t}})$ two process calculi equipped with behavioral equivalences $\asymp_{\mathrm{s}}$ and $\asymp_{\mathrm{t}}$, and transition relations $\longrightarrow_{\mathrm{s}}$ and $\longrightarrow_{\mathrm{t}}$, respectively. Let $[\![\cdot]\!] : \mathcal{S} \longmapsto \mathcal{T}$ be an encoding from $\mathcal{S}$ to $\mathcal{T}$. A formal definition of operational correspondence is the following:

Definition 4 (operational correspondence). *Given two process calculi $(\mathcal{S}, \asymp_{\mathrm{s}}, \longrightarrow_{\mathrm{s}})$ and $(\mathcal{T}, \asymp_{\mathrm{t}}, \longrightarrow_{\mathrm{t}})$, an encoding $[\![\cdot]\!] : \mathcal{S} \mapsto \mathcal{T}$ enjoys a (strong)* operational correspondence *if for each $S \in \mathcal{S}$ the following two properties holds:*

1. *If $S \longrightarrow_{\mathrm{s}} S'$ then $[\![S]\!] \longrightarrow_{\mathrm{t}} \asymp_{\mathrm{t}} [\![S']\!]$.*
2. *If $[\![S]\!] \longrightarrow_{\mathrm{t}} T$ then there is S' such that $S \longrightarrow_{\mathrm{s}} S'$ and $T \asymp_{\mathrm{t}} [\![S']\!]$.*

Requirements 1 and 2 assert that all possible executions of S may be simulated, up to behavioral equivalence, by its translation, and vice-versa. A notion of *weak operational correspondence* can be easily derived from Definition 4 by simply replacing $\longrightarrow_{\mathrm{s}}$ and $\longrightarrow_{\mathrm{t}}$ with their reflexive transitive closure, in requirements 1 and 2.

Full abstraction has two parts: *soundness*, which says that the equivalence between the translations of two source terms implies that of the source terms themselves; and *completeness*, which says the converse. While soundness is a necessary property and can be usually derived from the operational correspondence, completeness is in general hard to achieve because it implies a strong relationship between source and target calculi.

Definition 5 (soundness, completeness, and full abstraction). *Let $(\mathcal{S}, \asymp_{\mathrm{s}}, \longrightarrow_{\mathrm{s}})$ and $(\mathcal{T}, \asymp_{\mathrm{t}} \longrightarrow_{\mathrm{t}})$ be two process calculi. An encoding $[\![\cdot]\!] : \mathcal{S} \longmapsto \mathcal{T}$ is* sound *if $[\![S_1]\!] \asymp_{\mathrm{t}} [\![S_2]\!]$ implies $S_1 \asymp_{\mathrm{s}} S_2$ for each $S_1, S_2 \in \mathcal{S}$; it is* complete *if $S_1 \asymp_{\mathrm{s}} S_2$ implies $[\![S_1]\!] \asymp_{\mathrm{t}} [\![S_2]\!]$ for each $S_1, S_2 \in \mathcal{S}$; it is* fully-abstract *if it is sound and complete.*

Full abstraction will represent our correctness criterion for the encoding of polyadicity that we are going to present in the next section.

4 Encoding Polyadicity

In this section we give an encoding of polyadic Lπ into monadic Lπ. We present our encoding by comparison with Milner's encoding of polyadic synchronous π-calculus into monadic synchronous π-calculus [16]. Milner's idea is quite simple: One can emulate sending a tuple $\widetilde{b}$ by sending a fresh channel w along which all names b_i are transmitted sequentially. More precisely, Milner gives an encoding $\{\!|\cdot|\!\}$ from polyadic processes to monadic ones which is an homomorphism on all operators except input and output for which we have:

- $\{\!| a(x_1,\ldots,x_n).P |\!\} \stackrel{\text{def}}{=} a(w).\mathrm{INP}\langle w,x_1,\ldots,x_n\rangle.\{\!|P|\!\}$
- $\{\!|\overline{a}\langle b_1,\ldots,b_n\rangle.Q|\!\} \stackrel{\text{def}}{=} (\boldsymbol{\nu} w)\,(\overline{a}w.\mathrm{OUT}\langle w,b_1,\ldots,b_n\rangle.\{\!|Q|\!\})$

where

- $\mathrm{INP}\langle w,x_1,\ldots,x_n\rangle \stackrel{\text{def}}{=} w(x_1).w(x_2)\ldots w(x_n)$
- $\mathrm{OUT}\langle w,b_1,\ldots,b_n\rangle \stackrel{\text{def}}{=} \overline{w}b_1.\overline{w}b_2\ldots\overline{w}b_n$

and w is a fresh name, i.e., it is not free in the translated processes. Intuitively, $\mathrm{INP}\langle w,\widetilde{x}\rangle$ and $\mathrm{OUT}\langle w,\widetilde{b}\rangle$ model a protocol which takes care of instantiating each variable x_i with the correspondent name b_i by using a fresh channel w. Since w is private to $\mathrm{INP}\langle w,\widetilde{x}\rangle$ and $\mathrm{OUT}\langle w,\widetilde{b}\rangle$ no interferences are possible. It is easy to show that there is an *operational correspondence* between a polyadic process P and its translation $\{\!|P|\!\}$: (i) if $P\stackrel{\tau}{\longrightarrow}P'$ then $\{\!|P|\!\}\stackrel{\tau}{\longrightarrow}\gtrsim\{\!|P'|\!\}$ and (ii) if $\{\!|P|\!\}\stackrel{\tau}{\longrightarrow}P_1$ then there exists P' such that $P\stackrel{\tau}{\longrightarrow}P'$ and $P_1\gtrsim\{\!|P'|\!\}$ where $\lesssim$ is the *expansion relation* (see Definition 3). From the operational correspondence one can derive the *soundness* of the encoding when considering *barbed congruence* as the behavioral equivalence in both source and target languages. Unfortunately, as it is well-known, Milner's encoding is not complete and therefore it is not *fully-abstract*. As a counterexample take $R=a(\widetilde{x}).a(\widetilde{y}).\mathbf{0}$ and $S=a(\widetilde{x}).\mathbf{0}\mid a(\widetilde{y}).\mathbf{0}$; then R and S are barbed congruent but their encodings are not: $\{\!|S|\!\}$ may perform two consecutive inputs along a while in $\{\!|R|\!\}$ the input protocol $\mathrm{INP}\langle w,\widetilde{x}\rangle$ blocks the second input along a. In synchronous π-calculus, a similar counterexample can be given by using outputs instead of inputs. These counterexamples essentially say that Milner's encoding is not fully-abstract because the protocols $\mathrm{INP}\langle w,\widetilde{x}\rangle$ and $\mathrm{OUT}\langle w,\widetilde{b}\rangle$ prevent the continuations $\{\!|P|\!\}$ and $\{\!|Q|\!\}$ from evolving. Thus, one might think of adapting, somehow, Milner's encoding so that the protocols $\mathrm{INP}\langle w,\widetilde{x}\rangle$ and $\mathrm{OUT}\langle w,\widetilde{b}\rangle$ (or a variant of them) are in parallel with the continuations and not in sequence. In (full) π-calculus, such an adaptation is not possible because of the binding nature of the input prefix. This problem can be avoided in Lπ by relying on Lemma 1 which gives, under certain hypotheses, an interesting encoding for the substitution operator. Recall the definition of $\widetilde{a}\triangleright\widetilde{b}$ from Section 2.

Lemma 1 (Merro and Sangiorgi [15]). *Let $\widetilde{a}$ and $\widetilde{b}$ be two tuples with the same arity and such that $\widetilde{a} \cap \widetilde{b} = \emptyset$, and let P be an L$\widetilde{\pi}$-process such that all names $a_i \in \widetilde{a}$ do not appear free in input position in P. It holds that $(\boldsymbol{\nu}\widetilde{a})\,(\widetilde{a} \triangleright \widetilde{b} \mid P) \cong_{\mathrm{L}\widetilde{\pi}} P\{\widetilde{b}/\widetilde{a}\}$.*

In the following we show how Lemma 1 can be used to define an encoding $(\!|\cdot|\!)$ of polyadic Lπ-processes into monadic ones. For simplicity, we restrict ourselves to processes transmitting pairs of names. The general case, when tuples of arbitrary size are transmitted, can be derived straightforwardly. The encoding $(\!|\cdot|\!)$ is an homomorphism on all operators except input and output, for which we have:

- $(\!|\, a(\widetilde{x}).\, P \,|\!) \stackrel{\text{def}}{=} a(w).\, (\boldsymbol{\nu}\widetilde{x})\, (\mathrm{INP}\langle w, \widetilde{x}\rangle \mid (\!|\, P \,|\!))$
- $(\!|\, \overline{a}\langle\widetilde{b}\rangle \,|\!) \stackrel{\text{def}}{=} (\boldsymbol{\nu}w)\, (\overline{a}w \mid \mathrm{OUT}\langle w, \widetilde{b}\rangle)$

where $w \notin \mathrm{fn}((\!|\, P \,|\!))$, and supposing $\widetilde{x} = (x_1, x_2)$, $\widetilde{y} = (y_1, y_2)$, $\widetilde{b} = (b_1, b_2)$, and $\widetilde{x} \triangleright \widetilde{y} = !x_1(z).\, \overline{y_1}z \mid !x_2(z).\, \overline{y_2}z$ we define

- $\mathrm{INP}\langle w, \widetilde{x}\rangle \stackrel{\text{def}}{=} (\boldsymbol{\nu}c_1c_3)\, (\overline{w}c_1 \mid c_1(c_2).\, (\overline{c_2}c_3 \mid c_3(y_1).\, c_1(y_2).\, \widetilde{x} \triangleright \widetilde{y}))$
- $\mathrm{OUT}\langle w, \widetilde{b}\rangle \stackrel{\text{def}}{=} w(c_1).\, (\boldsymbol{\nu}c_2)\, (\overline{c_1}c_2 \mid c_2(c_3).\, (\overline{c_3}b_1 \mid \overline{c_1}b_2))$.

Like Milner's encoding, $(\!|\cdot|\!)$ is based on the send of a private channel w used by $\mathrm{INP}\langle w, \widetilde{x}\rangle$ and $\mathrm{OUT}\langle w, \widetilde{b}\rangle$ for transmitting names b_i. Unlike Milner's encoding, in $(\!|\cdot|\!)$ the send of names b_i produces n forwarders $x_i \triangleright b_i$ in parallel. More precisely, by Lemma 1, it holds that:

$$(\!|\, \overline{a}\langle\widetilde{b}\rangle \mid a(\widetilde{x}).\, P \,|\!) \xrightarrow{\tau} \gtrsim (\boldsymbol{\nu}\widetilde{x})\, (\widetilde{x} \triangleright \widetilde{b} \mid (\!|\, P \,|\!)) \cong_{\mathrm{L}\pi} (\!|\, P \,|\!)\{\widetilde{b}/\widetilde{x}\} \equiv (\!|\, P\{\widetilde{b}/\widetilde{x}\} \,|\!).$$

The encoding $(\!|\cdot|\!)$ is sound with respect to barbed congruence. Unfortunately, in this form, the encoding is not yet fully-abstract because it is not complete. As a counterexample take the processes $R = \overline{a}\langle\widetilde{b}\rangle$ and $S = (\boldsymbol{\nu}\widetilde{d})\, (\overline{a}\langle\widetilde{d}\rangle \mid \widetilde{d} \triangleright \widetilde{b})$, with $\widetilde{b} = (b_1, b_2)$ and $\widetilde{d} = (d_1, d_2)$; then $R \cong_{\mathrm{L}\widetilde{\pi}} S$ (see [15]) but $(\!|\, R \,|\!) \not\cong_{\mathrm{L}\pi} (\!|\, S \,|\!)$, indeed let $C[\cdot] = [\cdot] \mid T$ in which $T = a(w).\, (\boldsymbol{\nu}c_1c_3h)\, (\overline{w}c_1 \mid c_1(c_2).\, (\overline{c_2}c_3 \mid c_3(y_1).\, c_1(y_2).\, (\overline{y_1}h \mid h(x).\, \overline{m})))$, then $C[(\!|\, R \,|\!)] \not\Downarrow_m$ while $C[(\!|\, S \,|\!)] \Downarrow_m$. Notice that we may not find a similar counterexample by using two ground bisimilar processes R and S. This information allows us to give an amended variant of $(\!|\cdot|\!)$. By Theorem 2 we know that the encoding $[\![\cdot]\!]$ of Section 2 maps barbed congruent processes into ground bisimilar processes; that is, it holds that $P \cong_{\mathrm{L}\widetilde{\pi}} Q$ iff $[\![P]\!] \approx [\![Q]\!]$ (on image-finite processes). So, we can refine the encoding $(\!|\cdot|\!)$ by simply combining $(\!|\cdot|\!)$ with $[\![\cdot]\!]$. More precisely, we define an encoding $\langle\!|\![\cdot]\!|\!\rangle$ of L$\widetilde{\pi}$ into Lπ as the composition of $[\![\cdot]\!]$ and $(\!|\cdot|\!)$, thus if P is an L$\widetilde{\pi}$-process

$$\langle\!|\![P]\!|\!\rangle \stackrel{\text{def}}{=} (\!|\, [\![P]\!] \,|\!).$$

Notice that both encodings $[\![\cdot]\!]$ and $(\!|\cdot|\!)$ are *divergence-free*, that is, they do not introduce infinite internal computations, so also the encoding $\langle\!|\![\cdot]\!|\!\rangle$ does not introduce divergence.

5 Proving the Full Abstraction of $\langle\!\langle\![\cdot]\!]\rangle\!\rangle$

In this section we shall prove that, on image-finite and well-sorted processes, $\langle\!\langle\![\cdot]\!]\rangle\!\rangle$ is fully abstract with respect to barbed congruence. To this end we study the encoding $[\![(\!|\cdot|\!)]\!]$ obtained by inverting the order of the application of the encodings $[\![\cdot]\!]$ and $(\!|\cdot|\!)$. We first prove an operational correspondence, up to expansion, between processes P and $[\![(\!|P|\!)]\!]$. This will allows us to prove the soundness of $\langle\!\langle\![\cdot]\!]\rangle\!\rangle$. Then we derive the completeness of $\langle\!\langle\![\cdot]\!]\rangle\!\rangle$ from a completeness result for $[\![(\!|\cdot|\!)]\!]$.

Lemma 2 will allow us to prove the operational correspondence between processes P and $[\![(\!|P|\!)]\!]$.

Lemma 2 (Boreale [4]).

1. *Let $\widetilde{a}$, $\widetilde{b}$, $\widetilde{c}$ be tuples of names of the same size such that $(\widetilde{a} \cup \widetilde{c}) \cap \widetilde{b} = \emptyset$. Then $(\boldsymbol{\nu}\widetilde{b})\ (\widetilde{a} \to \widetilde{b} \mid \widetilde{b} \to \widetilde{c}) \gtrsim \widetilde{a} \to \widetilde{c}$.*
2. *Let P be an $\mathrm{L}\widetilde{\pi}$ process and $\widetilde{a}$ and $\widetilde{b}$ two tuples of names such that the names in $\widetilde{a}$ do not occur free in P in input-subject position and $\widetilde{a} \cap \widetilde{b} = \emptyset$. Then $(\boldsymbol{\nu}\widetilde{a})\ (\widetilde{a} \to \widetilde{b} \mid [\![P]\!]) \gtrsim [\![P]\!]\{\widetilde{b}/\widetilde{a}\}$.*

Remark 3. Lemma 2(2) can be seen as a variant of Lemma 1 up to $[\![\cdot]\!]$. Actually, Lemma 1 follows directly from Lemma 2(2) and Theorem 2(2).

Lemma 3. *Let P be a well-sorted process in $\mathrm{L}\widetilde{\pi}$ then:*

1. *Suppose that $P \xrightarrow{\alpha} P'$. Then we have:*
   (a) *if $\alpha = a(\widetilde{x})$ then $[\![(\!|P|\!)]\!] \xrightarrow{a(w)} \gtrsim (\boldsymbol{\nu}\widetilde{x})\ ([\![\mathrm{INP}\langle w, \widetilde{x}\rangle]\!] \mid [\![(\!|P'|\!)]\!])$;*
   (b) *if $\alpha = (\boldsymbol{\nu}\widetilde{c})\ \overline{a}\langle\widetilde{b}\rangle$, with $\widetilde{c}$ eventually empty, then $[\![(\!|P|\!)]\!] \xrightarrow{\overline{a}(p)} \gtrsim (\boldsymbol{\nu}\widetilde{c})\ ((\boldsymbol{\nu}w)\ (p \to w \mid [\![\mathrm{OUT}\langle w, \widetilde{b}\rangle]\!]) \mid [\![(\!|P'|\!)]\!])$, with $p \notin fn(P')$;*
   (c) *if $\alpha = \tau$ then $[\![(\!|P|\!)]\!] \xrightarrow{\tau} \gtrsim [\![(\!|P'|\!)]\!]$.*
2. *Suppose that $[\![(\!|P|\!)]\!] \xrightarrow{\alpha} P_1$. Then there exists $P' \in \mathrm{L}\widetilde{\pi}$ such that:*
   (a) *if $\alpha = a(w)$ then $P \xrightarrow{a(\widetilde{x})} P'$, for some $\widetilde{x}$, with $P_1 \gtrsim (\boldsymbol{\nu}\widetilde{x})\ ([\![\mathrm{INP}\langle w, \widetilde{x}\rangle]\!] \mid [\![(\!|P'|\!)]\!])$;*
   (b) *if $\alpha = \overline{a}(p)$ then $P \xrightarrow{(\boldsymbol{\nu}\widetilde{c})\ \overline{a}\langle\widetilde{b}\rangle} P'$, with $\widetilde{c}$ eventually empty, $p \notin fn(P')$ and $P_1 \gtrsim (\boldsymbol{\nu}\widetilde{c})\ ((\boldsymbol{\nu}w)\ (p \to w \mid [\![\mathrm{OUT}\langle w, \widetilde{b}\rangle]\!]) \mid [\![(\!|P'|\!)]\!])$;*
   (c) *if $\alpha = \tau$ then $P \xrightarrow{\tau} P'$ with $P_1 \gtrsim [\![(\!|P'|\!)]\!]$.*

Proof. By transition induction. The only subtle points arise in parts 1(c) and 2(c) where also Lemma 2 is used. Details can be found in [13].

Remark 4. Note that the lemma above is not true when considering ill-sorted processes. For instance, if $P = \overline{a}\langle b, c\rangle \mid a(x).\, Q$ then $[\![(\!|P|\!)]\!] \xrightarrow{\tau}$ while $P \not\xrightarrow{\tau}$.

From Lemma 3 we can derive a weak operational correspondence.

Lemma 4.

1. *If* $P \Longrightarrow P'$ *then* $[\![\langle\!| P |\!\rangle]\!] \Longrightarrow\!\gtrsim [\![\langle\!| P' |\!\rangle]\!]$;
2. *If* $[\![\langle\!| P |\!\rangle]\!] \Longrightarrow P_1$ *then there is* P' *s.t.* $P \Longrightarrow P'$ *and* $P_1 \gtrsim [\![\langle\!| P' |\!\rangle]\!]$;
3. $P \Downarrow_a$ *iff* $[\![\langle\!| P |\!\rangle]\!] \Downarrow_a$.

Proof. Parts 1 and 2 are proven by induction on the number of τ-moves by exploiting Lemma 3. Part 3 follows from parts 1 and 2, and Lemma 3.

Lemmas 3 and 4 allow us to prove the following two lemmas which will be useful for proving the soundness of $(\!|\cdot|\!)$.

Lemma 5. *Let* P *and* Q *be two* $\mathrm{L}\widetilde{\pi}$*-processes. Then*

$$[\![\langle\!| P |\!\rangle]\!] \mathrel{\dot{\approx}} [\![\langle\!| Q |\!\rangle]\!] \text{ implies } P \mathrel{\dot{\approx}} Q.$$

Proof. We use Lemmas 3 and 4 and the fact that $\lesssim \dot{\approx} \gtrsim \subset \dot{\approx}$ to prove that the relation $\mathcal{R} = \{(P, Q) : [\![\langle\!| P |\!\rangle]\!] \mathrel{\dot{\approx}} [\![\langle\!| Q |\!\rangle]\!]\}$ is a barbed bisimulation.

Lemma 6. *Let* P *and* Q *be two* $\mathrm{L}\widetilde{\pi}$*-processes. Then*

$$(\!| P |\!) \mathrel{\dot{\approx}} (\!| Q |\!) \text{ implies } P \mathrel{\dot{\approx}} Q.$$

Proof. Since $(\!|\cdot|\!) \stackrel{\text{def}}{=} \langle\!| [\![\cdot]\!] |\!\rangle$, by Theorem 1, we have $[\![\langle\!| [\![P]\!] |\!\rangle]\!] \mathrel{\dot{\approx}} [\![\langle\!| [\![Q]\!] |\!\rangle]\!]$. By Lemma 5 we have $[\![P]\!] \mathrel{\dot{\approx}} [\![Q]\!]$. By Theorem 1 we have $P \mathrel{\dot{\approx}} Q$.

The following lemma will allow us to prove the completeness of $(\!|\cdot|\!)$.

Lemma 7. *Let* P *and* Q *be two* $\mathrm{L}\widetilde{\pi}$*-processes. Then*

$$P \approx Q \text{ implies } [\![\langle\!| P |\!\rangle]\!] \approx [\![\langle\!| Q |\!\rangle]\!].$$

Proof. We prove that $\mathcal{S} = \{([\![\langle\!| P |\!\rangle]\!], [\![\langle\!| Q |\!\rangle]\!]) : P \approx Q\}$ is a ground bisimulation up to context and up to $\gtrsim$ [24]. Details can be found in [13].

Finally we prove that, on image-finite and well-sorted processes, the encoding $(\!|\cdot|\!)$ is fully-abstract with respect to barbed congruence.

Theorem 3 (full abstraction of $(\!|\cdot|\!)$). *Let* P *and* Q *be two image-finite and well-sorted processes in* $\mathrm{L}\widetilde{\pi}$*, then*

$$P \cong_{\mathrm{L}\widetilde{\pi}} Q \text{ iff } (\!| P |\!) \cong_{\mathrm{L}\pi} (\!| Q |\!).$$

Proof. The soundness follows from the compositionality of $[\![\cdot]\!]$ and $\langle\!| \cdot |\!\rangle$, and Lemma 6. As for completeness, by Theorem 2(1) we have $[\![P]\!] \approx [\![Q]\!]$. By Lemma 7 we have $[\![\langle\!| [\![P]\!] |\!\rangle]\!] \approx [\![\langle\!| [\![Q]\!] |\!\rangle]\!]$. By the monadic variant of Theorem 2(2) we have $\langle\!| [\![P]\!] |\!\rangle \cong_{\mathrm{L}\pi} \langle\!| [\![Q]\!] |\!\rangle$, i.e., $(\!| P |\!) \cong_{\mathrm{L}\pi} (\!| Q |\!)$.

6 What about $\langle\!|\cdot|\!\rangle$ and $[\![\langle\!|\cdot|\!\rangle]\!]$?

We have proved that the encoding $\langle\!|\cdot|\!\rangle$ is fully-abstract w.r.t. barbed congruence. Other possible candidates for a fully-abstract encoding of polyadic Lπ into monadic Lπ are: $\langle\!|\cdot|\!\rangle$ and $[\![\langle\!|\cdot|\!\rangle]\!]$. Unfortunately, none of them is fully-abstract w.r.t. barbed congruence or ground bisimilarity.

In Section 4 we already showed that $\langle\!|\cdot|\!\rangle$ is not fully-abstract w.r.t. barbed congruence. The encoding $\langle\!|\cdot|\!\rangle$ is not fully-abstract w.r.t. ground bisimilarity either. As a counterexample take $P = (\boldsymbol{\nu} a)\,(\overline{a}\langle\widetilde{b}\rangle \mid a(\widetilde{x}).\,\overline{c}\langle\widetilde{x}\rangle)$ and $Q = \overline{c}\langle\widetilde{b}\rangle$; then $P \approx Q$, but $\langle\!|\, P \,|\!\rangle \approx (\boldsymbol{\nu}\widetilde{x})\,(\widetilde{x} \triangleright \widetilde{b} \mid \langle\!|\, \overline{c}\langle\widetilde{x}\rangle \,|\!\rangle)$, and $\langle\!|\, Q \,|\!\rangle = \langle\!|\, \overline{c}\langle\widetilde{b}\rangle \,|\!\rangle$, and therefore $\langle\!|\, P \,|\!\rangle \not\approx \langle\!|\, Q \,|\!\rangle$.

By Lemma 7 the encoding $[\![\langle\!|\cdot|\!\rangle]\!]$ is complete w.r.t. ground bisimilarity. Since the encoding $[\![\langle\!|\cdot|\!\rangle]\!]$ enjoys an operational correspondence up to expansion (Lemma 3), one may hope that $[\![\langle\!|\cdot|\!\rangle]\!]$ is sound w.r.t. ground bisimilarity and therefore fully-abstract. Unfortunately, $[\![\langle\!|\cdot|\!\rangle]\!]$ is not sound w.r.t. ground bisimilarity. As a counterexample take $P = (\boldsymbol{\nu}\widetilde{c})\,\overline{a}\langle\widetilde{c}\rangle$ and $Q = (\boldsymbol{\nu}\widetilde{c})\,(\overline{a}\langle\widetilde{c}\rangle \mid \overline{c_1}\langle b\rangle)$, with $\widetilde{c} = (c_1, c_2)$; then $P \cong_{\mathrm{L}\widetilde{\pi}} Q$ (see [15]) and also $\langle\!|\, P \,|\!\rangle \cong_{\mathrm{L}\pi} \langle\!|\, Q \,|\!\rangle$, by Theorem 2 $[\![\langle\!|\, P \,|\!\rangle]\!] \approx [\![\langle\!|\, Q \,|\!\rangle]\!]$, but $P \not\approx Q$. The encoding $[\![\langle\!|\cdot|\!\rangle]\!]$ is not fully-abstract w.r.t. barbed congruence either. As a counterexample take the processes $P = \overline{a}\langle\widetilde{b}\rangle$ and $Q = (\boldsymbol{\nu}\widetilde{d})\,(\overline{a}\langle\widetilde{d}\rangle \mid \widetilde{d} \triangleright \widetilde{b})$, with $\widetilde{b} = (b_1, b_2)$ and $\widetilde{d} = (d_1, d_2)$; then, as already shown in Section 4, $P \cong_{\mathrm{L}\widetilde{\pi}} Q$ and $\langle\!|\, P \,|\!\rangle \not\cong_{\mathrm{L}\pi} \langle\!|\, Q \,|\!\rangle$; since the encoding $[\![\cdot]\!]$ is sound w.r.t. barbed congruence (which follows by Theorem 1 and the compositionality of $[\![\cdot]\!]$) we have that $[\![\langle\!|\, P \,|\!\rangle]\!] \not\cong_{\mathrm{L}\pi} [\![\langle\!|\, Q \,|\!\rangle]\!]$.

7 An Encoding of Polyadicity in Calculi with Non-binding Input Prefix

In Section 4, we said that Milner's encoding is not fully-abstract because the protocols $\mathrm{INP}\langle w, \widetilde{x}\rangle$ and $\mathrm{OUT}\langle w, \widetilde{b}\rangle$ prevent the continuations $\{\!|\, P \,|\!\}$ and $\{\!|\, Q \,|\!\}$ to evolve. Actually, the real problem is the binding nature of the input prefix. Indeed, we can easily change the encoding of $\overline{a}\langle\widetilde{b}\rangle.\, P$ by putting the protocol $\mathrm{OUT}\langle w, \widetilde{b}\rangle$ in parallel with the continuation but we cannot do the same with the encoding of input prefixes.

On the contrary, in calculi where the input prefix is non-binding, such as *Chi calculus* [9], *Fusion calculus* [19] and πF-*calculus* [10], we can adapt Milner's encoding by simply putting the protocol $\mathrm{INP}\langle w, \widetilde{x}\rangle$ in parallel with the continuation. We conjecture that, in these calculi, such a variant of Milner's encoding is fully-abstract.

Let us consider, for instance, the Fusion calculus. [1] For our purposes it suffices to consider a finite fragment. The extension of our encoding when infinite processes are allowed is straightforward. The grammar of finite Fusion calculus has operators of inaction, *non-binding input prefix*, output prefixing, parallel composition, and restriction:

[1] Actually, it might be easier to work with the πF-*calculus*. We consider the Fusion calculus and not πF only because the theory of Fusion is more stable.

$$P ::= \mathbf{0} \;\Big|\; a\langle\widetilde{x}\rangle.\, P \;\Big|\; \overline{a}\langle\widetilde{b}\rangle.\, P \;\Big|\; P \mid P \;\Big|\; (a)P$$

Conventions about names are as in π-calculus, except for the non-binding input prefix for which we have:

$$\mathrm{fn}(a\langle\widetilde{x}\rangle.\, P) \stackrel{\mathrm{def}}{=} \{a\} \cup \{x_1, \ldots, x_n\} \cup \mathrm{fn}(P) \;\text{ and }\; \mathrm{bn}(a\langle\widetilde{x}\rangle.\, P) \stackrel{\mathrm{def}}{=} \mathrm{bn}(P).$$

We write $(\widetilde{x})P$ for $(x_1)\ldots(x_n)P$. Conventions about processes and substitutions are as in π-calculus. The *reduction semantics* is defined by means of a notion of structural congruence (essentially the same as in π-calculus) and a reduction relation. For simplicity, we give the basic reduction rules for the monadic calculus, the generalization to the polyadic case is slightly more complex:

$$\texttt{(comm1)}: (x)(P \mid a\langle x\rangle.\, Q \mid \overline{a}\langle y\rangle.\, R) \longrightarrow (x)((P \mid Q \mid R)\{y/x\})$$

$$\texttt{(comm2)}: (y)(P \mid a\langle x\rangle.\, Q \mid \overline{a}\langle y\rangle.\, R) \longrightarrow (y)((P \mid Q \mid R)\{x/y\})$$

Notice that the restrictions (x) and (y) in the derivatives make sense only when $x = y$, otherwise, up to structural congruence, they disappear. The definitions of observability, barbed bisimilarity, and barbed congruence are essentially the same as in π-calculus. Finally, an important derived process, called fusion, can be defined as follows: $\{\widetilde{x} = \widetilde{y}\} \stackrel{\mathrm{def}}{=} (u)(\overline{u}\langle\widetilde{x}\rangle.\, \mathbf{0} \mid u\langle\widetilde{y}\rangle.\, \mathbf{0})$.

In Fusion calculus, communications can arise only in the presence of a scoping construct delimiting their effects (see reduction rules `(comm1)` and `(comm2)`). So, to observe all potential communications (and their effects) it makes sense to consider a notion of barbed congruence obtained by closing barbed bisimulation under contexts that bind all free names of the tested processes. Similar *closing contexts* have been used in typed calculi [26]. As in the definition of *testing equivalences* [7], these contexts signal success by emitting along names that do not appear in the tested processes.

Definition 6 (closed barbed congruence). *Two processes P and Q are* closed barbed congruent*, written $\cong_c$, if for each context $C[\cdot]$ such that* $\mathrm{fn}(P) \cap \mathrm{fn}(C[P]) = \mathrm{fn}(Q) \cap \mathrm{fn}(C[Q]) = \emptyset$*, it holds that* $C[P] \;\dot{\approx}\; C[Q]$.

It is immediate to adapt this definition to other calculi, such as π-calculus. Actually, in π-calculus and CCS, closed barbed congruence coincides with barbed congruence: One can prove that closed barbed congruence coincides with the closure under substitutions of early bisimulation, by adapting the proofs in [22, 2]; since the closure under substitutions of early bisimulation is known to coincide with barbed congruence, the two definitions of barbed congruence coincide. Unfortunately, standard and closed barbed congruence do not coincide in Fusion.

Milner's encoding can be rewritten in Fusion calculus so that the instantiation of names does not block the continuations:

$$(\!|\, a\langle x_1, \ldots, x_n\rangle.\, P \,|\!) \stackrel{\text{def}}{=} (w)aw.\, (wx_1.\, wx_2. \ldots . wx_n \mid (\!|\, P \,|\!))$$

$$(\!|\, \overline{a}\langle b_1, \ldots, b_n\rangle.\, Q \,|\!) \stackrel{\text{def}}{=} (w)\overline{a}w.\, (\overline{w}b_1.\, \overline{w}b_2. \ldots . \overline{w}b_n \mid (\!|\, Q \,|\!))$$

Note that the continuations $(\!|\, P \,|\!)$ and $(\!|\, Q \,|\!)$ may evolve without waiting for the n communications along the private channel w. This encoding is not sound w.r.t. standard barbed congruence and not even w.r.t. *hyperequivalence* [19], because, in some sense, the encoding breaks the preemptive power of fusions. More precisely, if we take $R = \{a = b\} \mid \{c = d\}$ and $S = \{a = b\}.\, \{c = d\}$[2], then R and S are not equivalent while their translations are. Nevertheless, we believe that the encoding $(\!|\, \cdot \,|\!)$ is fully abstract with respect to closed barbed congruence. Our conjecture is due to the fact that closed barbed congruence is insensitive to fusion prefixing, that is, it handles fusion actions as silent moves. As a consequence, the counterexample above is not valid anymore because processes $\{a = b\} \mid \{c = d\}$ and $\{a = b\}.\, \{c = d\}$ are closed barbed congruent. Unfortunately, we cannot prove the full abstraction of $(\!|\, \cdot \,|\!)$ by using the same proof techniques of Section 5 because we do not know yet a labeled characterization of closed barbed congruence in Fusion.

8 Conclusions and Related Works

We have presented a *divergence-free* encoding $[\![\cdot]\!]$ of polyadic Lπ into monadic Lπ inspired by Milner's encoding. The encoding exploits a property of Lπ saying that, under certain hypotheses, substitution can be encoded in terms of links, restriction, and parallel composition. This property allows us to define an encoding of polyadicity where the machinery emulating the transmission of tuples does not block continuations.

We have proved that, on image-finite and well-sorted processes, $[\![\cdot]\!]$ is fully-abstract with respect to barbed congruence. This shows that in Lπ (i) polyadicity does not add extra expressive power, and (ii) when studying the theory of polyadic Lπ we can focus on the simpler monadic variant. Finally, we have proposed an encoding of polyadicity in name-passing calculi with non-binding input prefix, such as *Chi*, *Fusion* and πF *calculi* [9, 19, 10], which is based on the same idea of $[\![\cdot]\!]$.

Of course, our encodings (as the Milner's one) do not preserve well-sortedness. This is a minor point because in monadic calculi there cannot be arity mismatching.

Note that we have used synchronous barbed congruence where both input and output actions are observed. Sometimes, in asynchronous calculi, only output barbs are taken into account validating the law $a(\widetilde{x}).\, \overline{a}\langle\widetilde{x}\rangle = \mathbf{0}$. This law may be questioned because it introduces divergences. For instance, from $a(\widetilde{x}).\, \overline{a}\langle\widetilde{x}\rangle = \mathbf{0}$ we can derive the equality $!a(\widetilde{x}).\, \overline{a}\langle\widetilde{x}\rangle \mid \overline{a}b = \overline{a}b$ between a divergent and a non-divergent process. Our encoding is not complete w.r.t. asynchronous barbed

[2] A formal definition for S is $(uz)(ua.\, zc \mid \overline{u}b.\, \overline{z}d)$

congruence precisely because $(\!|a(\widetilde{x}).\,\overline{a}\langle\widetilde{x}\rangle|\!)$ and $(\!|\mathbf{0}|\!)$ are not asynchronous barbed congruent. However, we believe that $(\!|\cdot|\!)$ is fully-abstract w.r.t. a variant of asynchronous barbed congruence which is sensitive to divergence, along the lines of [28].

The works which are most closely related to ours are [29, 21] where type systems for monadic π-processes are introduced in order to capture the communication protocol underlying Milner's encoding. More precisely, in [29] a notion of graph type is introduced and studied. Nodes of a graph type represent atomic actions, and edges an activation ordering between them. The approach in [21] is similar but the type system is simpler. Both papers show a full abstraction result with respect to typed contextual equivalences that reject all contexts which do not respect the protocol imposed by the encoding. While [29, 21] work on the full π-calculus our result only applies in Lπ. This is because Lemmas 1 and 2 only work on Lπ-processes. On the other hand, we prove a sharper result because we get the completeness of the encoding with respect to all monadic contexts without rejecting "hostile" contexts.
In [8], Fournet and Gonthier provide, among other results, a fully-abstract encoding of polyadic into monadic Join Calculus. Apart from the differences among the two process calculi, the encoding in [8] is technically quite different from ours; for instance, as the authors themselves say, their translation encodes and then subsequently decodes tuples twice.

Acknowledgments A series of discussions with Davide Sangiorgi has inspired the work reported in the paper. I thank Paola Quaglia, David Walker and Björn Victor for insightful discussions on the encoding of polyadicity in π and Fusion calculi. I thank Ilaria Castellani, Silvano Dal-Zilio, Christine Röckl, and the anonymous referees for useful comments on the paper.

References

[1] R. Amadio. An asynchronous model of locality, failure, and process mobility. In *Proc. Coordination'97*, LNCS 1282, Springer Verlag, 1997.
[2] R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195:291–324, 1998.
[3] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29:737–760, 1992.
[4] M. Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theoretical Computer Science*, 195:205–226, 1998.
[5] G. Boudol. Asynchrony and the π-calculus. Technical Report RR-1702, INRIA-Sophia Antipolis, 1992.
[6] G. Boudol. The pi-calculus in direct style. In *Proc. 24th POPL*. ACM Press, 1997.
[7] R. De Nicola and R. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
[8] C. Fournet and G. Gonthier. The Reflexive Chemical Abstract Machine and the Join calculus. In *Proc. 23th POPL*. ACM Press, 1996.
[9] Y. Fu. A proof theoretical approach to communication. In *24th ICALP*, volume 1256 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[10] P. Gardner and L. Wischik. The πF-calculus: a π-calculus with fusions. Personal Communication, 1999.
[11] K. Honda and M. Tokoro. An Object Calculus for Asynchronous Communication. In Proc. *ECOOP'91,* LNCS 512, Springer Verlag, 1991.
[12] K. Honda and M. Tokoro. A Small Calculus for Concurrent Objects. In *OOPS Messanger*, Association for Computing Machinery. 2(2):50-54, 1991.
[13] M. Merro. Locality and polyadicity in asynchronous name-passing calculi. Available at *http://www-sop.inria.fr/meije/personnel/Massimo.Merro.html.*, 1999.
[14] M. Merro, H. Hüttel, J. Kleist, and U. Nestmann. Migrating Objects as Mobile Processes. To appear as a INRIA/BRICS Technical Report, 1999.
[15] M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. In *25th ICALP*, volume 1443 of *Lecture Notes in Computer Science*. Springer Verlag, 1998. The full-paper will appear as an INRIA Technical report.
[16] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, LFCS, Dept. of Comp. Sci., Edinburgh Univ., October 1991.
[17] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.
[18] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *19th ICALP*, LNCS 623, Springer Verlag, 1992.
[19] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proc. LICS'98*, IEEE Computer Society Press., 1998.
[20] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. To appear in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press.
[21] P. Quaglia and Walker D.. On encoding pπ in mπ. In *Proc. FST & TCS*, volume 1530 of *Lecture Notes in Computer Science*, pages 42–53. Springer Verlag, 1998.
[22] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, University of Edinburgh, 1992.
[23] D. Sangiorgi. Lazy functions and mobile processes. Technical Report RR-2515, INRIA-Sophia Antipolis, 1995.
[24] D. Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. *Theoretical Computer Science*, 155:39–83, 1996.
[25] D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(2):235–274, 1996.
[26] D. Sangiorgi. The name discipline of receptiveness. In *24th ICALP*, volume 1256 of *Lecture Notes in Computer Science*. Springer Verlag, 1997. To appear in TCS.
[27] D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". In *Proc. CONCUR '92*, LNS 630, Springer Verlag, 1992.
[28] D. Walker. Bisimulation and divergence. *Information and Computation*, 85(2):202–241, 1990.
[29] N. Yoshida. Graph types for monadic mobile processes. In *Proc. FST & TCS*, LCNS 1180, Springer Verlag, 1996.
[30] N. Yoshida. Minimality and Separation Results on Asynchronous Mobile Processes: representability theorem by concurrent combinators. In *9th CONCUR*, LNCS 1466, Springer Verlag, 1998.