Technical Report 06/2005, Dip. di Informatica, Univ. di Roma “La Sapienza”

On the Relative Expressive Power of Asynchronous Communication Primitives

Daniele Gorla
Dipartimento di Informatica
Università di Roma “La Sapienza”

Abstract

In this paper, we study eight asynchronous communication primitives, arising from the combination of three features: *arity* (monadic vs polyadic data), *communication medium* (message passing vs shared dataspaces) and *pattern-matching*. Each primitive has been already used in at least one language appeared in literature; however, to uniformly reason on such primitives, we plugged them in a common framework inspired by the asynchronous π-calculus. By means of possibility/impossibility of ‘reasonable’ encodings, we compare every pair of primitives to obtain a hierarchy of languages based on their relative expressive power.

1 Introduction

In the last 25 years, several languages and formalisms for distributed and concurrent systems appeared in literature. Some of them (e.g., CCS [21] and the π-calculus [26]) are mostly mathematical models, mainly used to formally reason on concurrent systems; other ones (e.g., LINDA [17]) are closer to actual programming languages and are mainly focused on issues like usability and flexibility. As a consequence, the former ones are usually very essential, while the latter ones provide more sophisticated and powerful programming constructs.

Despite their differences, there are, however, some basic features that are somewhat implemented in all these languages. Roughly speaking, these features can be described as the possibility of having different *execution threads* (or *processes*) that *run concurrently* by interacting via some form of *communication*. At least at a first glance, the last feature (i.e., the inter-process communication) has yield the highest variety of proposals. These arose from the possibility of having synchronous/asynchronous primitives, monadic/polyadic data, first-order/higher-order values, dataspace-based/channel-based communication media, local/remote exchanges (whenever processes are explicitly distributed, like in [10, 13]), built-in pattern-matching mechanisms, point-to-point/broadcasting primitives, and so on. The aim of this work is to formally study some of these proposals and to organise them in a clear hierarchy, based on their expressive power. Hopefully, our results should help to understand the peculiarities of ev-

ery communication primitive and, as a consequence, they could be exploited to choose the 'right' primitive when designing new languages and formalisms.

We focus on *asynchronous* communication primitives, since they are the most basilar ones. Among the remaining features mentioned above, we focus on *arity of data*, *communication medium* and possibility of *pattern-matching*. The expressiveness of the omitted features has been already dealt with elsewhere [29, 13, 15]; we leave as a future work the integration of these results in our framework. Notice that we studied pattern-matching because it is nowadays becoming more and more important, especially in languages that deal with complex data like XML [1, 5, 11]. However, for the sake of simplicity, we consider here a very basic form of pattern-matching, that only checks for name equality while retrieving a datum; the formal study of more flexible and powerful mechanisms (e.g., those in [14]) is left for future work.

By combining the three features chosen, we obtain eight communication primitives that have been all already employed elsewhere, e.g. in [20, 4, 17, 10, 13, 11]. However, to uniformly reason on such primitives, we plugged them in a common framework inspired by the asynchronous π-calculus; we choose the π-calculus because nowadays it is one of the best-established workbenches for theoretical reasoning on concurrent systems. By following [30, 12, 25], we shall compare the resulting languages by means of their *relative expressive power*, i.e. we shall try to encode one in the other and study the properties of the encoding function. More precisely, we shall exploit 'reasonable' encodings (as introduced in [25]), or impossibility of such encodings, to compare every pair of primitives, thus obtaining a hierarchy of languages based on their relative expressive power.

1.1 On Assessing the Expressiveness of a Language

To study the expressive power of a programming language, several techniques can be exploited. A first, very rough, test is to determine whether a language is Turing complete or not; however, since almost all 'usefull' languages are Turing complete, this criterion is too coarse to compare different languages.

A second, more informative, approach to show that language $\mathcal{L}_1$ is more expressive than language $\mathcal{L}_2$ is to find a problem that can be solved in $\mathcal{L}_1$ under some conditions that cannot be met by any solution in $\mathcal{L}_2$. For example, this technique can be very naturally used to prove that non-deterministic Turing machines are more powerful than deterministic ones. Indeed, the former machines can solve all problems in NP with a polynomial-time computation, whereas it is very unlikely that the latter machines can satisfy this property (because of the very well-known conjecture "P $\neq$ NP").

Another interesting approach to show that $\mathcal{L}_1$ is at least as expressive as $\mathcal{L}_2$ consists in encoding $\mathcal{L}_2$ in $\mathcal{L}_1$ (where an encoding is a function that translates every $\mathcal{L}_2$-term in a $\mathcal{L}_1$-term) and studying the properties of this encoding. This is the approach we shall follow in this paper. Before discussing the properties one may require for an encoding, let us remark that this approach is very appealing for at least two reasons. First, it is a natural way to show how the key features of $\mathcal{L}_2$ can be rendered in $\mathcal{L}_1$. Second, it allows us to also carry out quantitative measures on language expressiveness: we can consider aspects like the size and the complexity of the encoding of a $\mathcal{L}_2$-term w.r.t. the source term and, consequently, quantitatively assess the encoding proposed.

Of course, the encoding function must preserve the 'essence' of the translated term, i.e. to be meaningful an encoding should not change the functionalities and the behaviours of source terms. This requirement can be formalised in different ways. A first possibility (usually called *semantical equivalence*) is to fix an equivalence, say $\sim$, and require that the encoding maps every $\mathcal{L}_2$-term into a $\sim$-equivalent $\mathcal{L}_1$-term. According to the discriminating power of $\sim$, this requirement can be too demanding. Moreover, this property can only be investigated when $\mathcal{L}_1$ and $\mathcal{L}_2$ are very similar, i.e. whenever they share some notion of equivalence. This property can be weakened by choosing an abstract semantic theory $\mathcal{S}$ and considering the equivalences generated in $\mathcal{L}_1$ and $\mathcal{L}_2$ by $\mathcal{S}$, say $\sim_1^{\mathcal{S}}$ and $\sim_2^{\mathcal{S}}$. Then, the so called *full abstraction* property requires that the encoding respects $\mathcal{S}$, i.e. it maps $\sim_2^{\mathcal{S}}$-equivalent terms into $\sim_1^{\mathcal{S}}$-equivalent terms and vice versa.

Semantical equivalence and full abstraction are both defined w.r.t. a fixed notion of equivalence (viz., $\sim$ or $\mathcal{S}$). In concurrency, we have an incredibly wide range of equivalences; thus, fixing one or another is highly debatable. This is even more dramatic when proving impossibility results, that are crucial to build up a strict hierarchy of languages: every separation result based on a fixed semantic theory could be criticised by saying that it actually compares not the expressive power of the languages, but the discriminating power of the semantic theories. To prove that $\mathcal{L}_1$ is strictly more powerful than $\mathcal{L}_2$ it is better to fix a set of minimal properties that every encoding should satisfy and prove that no such encoding of $\mathcal{L}_2$ in $\mathcal{L}_1$ exists. We shall formally present the minimal properties we are going to exploit in this paper later on; for the moment, we only want to discuss and justify the most debatable ones.

Homomorfism w.r.t. parallel composition A first requirement (taken from [12, 25]) is that the encoding, $[\![\cdot]\!]$, should not reduce the degree of parallelism present in the source term. This property, called *homomorfism w.r.t. parallel composition*, can be formalised as

$$[\![T \mid_2 T']\!] = [\![T]\!] \mid_1 [\![T']\!] \qquad (\dagger)$$

where '$\mid_i$' is the parallel composition operator in language $\mathcal{L}_i$. Property $(\dagger)$ implies that, to be more expressive than $\mathcal{L}_2$, a (concurrent) language $\mathcal{L}_1$ must implement a certain behaviour with at least the same degree of distribution. Indeed, suppose that a centralised entity is needed to translate a certain term $T \mid_2 T'$; this means that $\mathcal{L}_1$'s constructs are not powerful enough to let parallel processes organise and simulate the behaviour of $T \mid_2 T'$ autonomously. The possibility of handling a high number of parallel components without any centralised entity is a clear evidence of the expressive power of the language constructs.

Clearly, a simple consequence of $(\dagger)$ is that no sequential language $\mathcal{L}_1$ can be more expressive than a concurrent language $\mathcal{L}_2$. However, the reader should not believe that this fact undermines the current implementations of some concurrent languages (take, e.g., the implementation of PICT [27], a programming language based on the π-calculus). Property $(\dagger)$ simply implies that concurrent languages are more powerful than sequential ones, a claim that is hardly debatable. Not incidentally, several problems arise when passing from a sequential setting to a concurrent one, because of the

enhanced possibilities of interactions (mutual exclusion, deadlock, synchronisation, ...).

Another possible critique to (†) is that there exist 'good' encodings that do not map '$|_2$' in '$|_1$'; for example, in the setting of CCS [21], take the encoding of CCS into its parallel-free fragment that implements '$|$' via '+' (the non-deterministic choice), as described by the expansion theorem:

$$\begin{array}{rcl} [\![a.P \mid b.Q]\!] & \triangleq & a.[\![P \mid b.Q]\!] + b.[\![a.P \mid Q]\!] \\ [\![a.P \mid \bar{a}.Q]\!] & \triangleq & a.[\![P \mid \bar{a}.Q]\!] + \bar{a}.[\![a.P \mid Q]\!] + \tau.[\![P \mid Q]\!] \end{array}$$

This encoding enjoys semantical equivalence w.r.t. strong bisimulation, a really strong and rare property. Neveretheless, we can still vindicate the validity of (†) by saying that such an encoding is not fully satisfactory for quantitative reasons, since it leads to an exponential blow-up of the process syntax.

Divergence freedom Another property we shall require to our encodings is the impossibility of introducing divergence, i.e. every non-terminating computation in the encoded term must correspond to a non-terminating computation of the source term. This property, called *divergence freedom*, is formalised as

$$[\![T]\!] \Uparrow \quad \text{iff} \quad T \Uparrow \qquad (\ddagger)$$

where '$\Uparrow$' denotes the presence of a non-terminating computation. Clearly, a terminating term must be mapped in a terminating term, otherwise the functionalities of the source term would be changed by the encoding. Indeed, if T is a $\mathcal{L}_2$-term that always halts and $[\![T]\!]$ exhibits a non-terminating computation, T's behaviour has been modified by $[\![\cdot]\!]$.

Again, also (‡) can be considered too strong. Indeed, one may argue that divergence does not matter when it arises with negligible probability, as in practice it is very unlikely to observe it. While this observation makes sense from an implementative point of view, it seems quite clumsy from a theoretical point of view. Indeed, take a terminating $\mathcal{L}_2$-term and suppose that every encoding of $\mathcal{L}_2$ in $\mathcal{L}_1$ introduces divergence (even with negligible probability); this can be used as an evidence of the fact that the constructs of $\mathcal{L}_1$ are not powerful enough to mimick the constructs of $\mathcal{L}_2$: indeed, to preserve all the functionalities of the source term, every encoding has to add further behaviours to the encoded term. Thus, $\mathcal{L}_1$ cannot be as expressive as $\mathcal{L}_2$. At most, the probability of having a divergent encoding from a non-divergent term can be considered as a measure of the gap between the expressive power of the languages.

A similar critique to (‡) could be that divergence only matters if it arises in fair computations. Again, such a refinement of (‡) is really meaningful when considering the encoding as an implementation of the source language in the target one. From the expressiveness prespective, the fact that all encodings introduce (unfair) divergence when passing from $\mathcal{L}_2$ to $\mathcal{L}_1$ is an evidence of the irriducibility of $\mathcal{L}_2$ to $\mathcal{L}_1$.

1.2 Overview of the Paper

Our results show that the communication paradigm underlying LINDA [17] (polyadic, dataspace-based and with pattern-matching) is at the top of the hierarchy; not inci-

dentally, LINDA's paradigm has been used in actual programming languages [3, 16]. On the opposite extreme, we have the communication paradigm used in Ambient [10] (monadic, dataspace-based but without pattern-matching). Such a paradigm is very simple but also very poor; indeed, Ambient's expressive power mostly arises from the mobility primitives. Strictly in the middle, we find the asynchronous π-calculus (channel-based and without pattern-matching), in its monadic and polyadic version. This result stresses the fact that the π-calculus is a good compromise between expressiveness and simplicity. As a further contribution, we also prove that the polyadic π-calculus is strictly more expressive than the monadic one. A posteriori, this fact justifies the use of type-systems [22, 31, 28] to obtain a fragment of the former calculus that can be reasonably translated in the latter one.

The paper is organised as follows. In Section 2, we present a family of eight π-based asynchronous calculi arising from the combination of the three features studied. In Section 3, we present the criteria an encoding should satisfy to be a reasonable means for language comparison; there, we also sum-up the results of the paper, that are proved in Sections 4 and 5. We start with the encodability results and then we present the impossibility results, that are the main contribution of our work. Finally, in Section 6, we conclude the paper by also touching upon related work.

This paper is an extended version of [18]. With respect to the short version, we give here more details on some technical proofs and we formally prove Turing completeness of the languages studied. Moreover, we added Section 1.1 to better justify the approach followed in assessing language expressiveness.

2 A Family of π-based Calculi

As we said in the Introduction, we shall assess the expressiveness of the communication primitives studied by putting them in a common framework, inspired by the asynchronous π-calculus. We assume two disjoint and countable sets: *names*, $\mathcal{N}$, ranged over by $a, b, x, y, n, m, \cdots$, and *process variables*, $\mathcal{X}$, ranged over by $X, Y, \cdots$. Notationally, when a name is used as a channel, we shall prefer letters $a, b, c, \cdots$; when a name is used as an input variable, we shall prefer letters $x, y, z, \cdots$; to denote a generic name, we shall use letters $n, m, \cdots$. The (parametric) syntax of our calculi is

$$\begin{array}{rcl} P, Q, R & ::= & \mathbf{0} \mid OUT \mid IN.P \mid (\nu n)P \mid P|Q \\ & \mid & \mathbf{if}\ n = m\ \mathbf{then}\ P\ \mathbf{else}\ Q \mid \mathbf{rec}\, X.P \mid X \end{array}$$

The different calculi will be obtained by plugging into this basic syntax a proper definition for input (IN) and output (OUT) actions. As usual, $\mathbf{0}$ and $P|Q$ denote the terminated process and the parallel composition of two processes, while $(\nu n)P$ restricts to P the visibility of n; finally, $\mathbf{if}\ n = m\ \mathbf{then}\ P\ \mathbf{else}\ Q$, $\mathbf{rec}\, X.P$ and X are the standard constructs for conditional evolution, recursive process definition and process invocation. Notationally, $\mathbf{if}\ n = m\ \mathbf{then}\ P$ denotes a conditional construct with a terminated else-branch; moreover, we shall omit trailing occurrences of $\mathbf{0}$.

In this paper, we study the possible combinations of three features for asynchronous communications: *arity* (monadic vs. polyadic data), *communication medium* (channels

vs. shared dataspaces) and *pattern-matching*. As a result, we have a family of eight calculi, denoted as $\Pi_{a,m,p}$, whose generic element is denoted as $\pi_{\beta_1\beta_2\beta_3}$, where $\beta_i \in \{0,1\}$. Intuitively, $\beta_1 = 1$ iff we have polyadic data, $\beta_2 = 1$ iff we have channel-based communications and $\beta_3 = 1$ iff we have pattern-matching. Thus, the full syntax of every calculus is obtained from the following productions:

$$
\begin{array}{llll}
\pi_{000}: & P,Q,R ::= \ldots & IN ::= (x) & OUT ::= \langle b \rangle \\
\pi_{001}: & P,Q,R ::= \ldots & IN ::= (T) & OUT ::= \langle b \rangle \\
\pi_{010}: & P,Q,R ::= \ldots & IN ::= a(x) & OUT ::= \overline{a}\langle b \rangle \\
\pi_{011}: & P,Q,R ::= \ldots & IN ::= a(T) & OUT ::= \overline{a}\langle b \rangle \\
\pi_{100}: & P,Q,R ::= \ldots & IN ::= (\widetilde{x}) & OUT ::= \langle \widetilde{b} \rangle \\
\pi_{101}: & P,Q,R ::= \ldots & IN ::= (\widetilde{T}) & OUT ::= \langle \widetilde{b} \rangle \\
\pi_{110}: & P,Q,R ::= \ldots & IN ::= a(\widetilde{x}) & OUT ::= \overline{a}\langle \widetilde{b} \rangle \\
\pi_{111}: & P,Q,R ::= \ldots & IN ::= a(\widetilde{T}) & OUT ::= \overline{a}\langle \widetilde{b} \rangle
\end{array}
$$

where

$$T ::= x \mid \ulcorner n \urcorner \qquad \text{(\textit{Template})}$$

and $\widetilde{\cdot}$ denotes a (possibly empty) sequence of elements of kind $\cdot$ (whenever useful, we shall write a tuple $\widetilde{\cdot}$ as the sequence of its elements, separated by a comma). Template fields of kind x are called *formal* and can be replaced by every name upon withdrawal of a datum; template fields of kind $\ulcorner n \urcorner$ are called *actual* and impose that the datum withdrawn contains exactly name n. As usual, $a(\cdots, x, \cdots).P$ and $(\nu x)P$ bind x in P, while $\mathbf{rec}\, X.P$ binds X in P. The corresponding notions of free and bound names of a process, $\text{FN}(P)$ and $\text{BN}(P)$, and of alpha-conversion, $=_\alpha$, are assumed. We let $\text{N}(P)$ denote $\text{FN}(P) \cup \text{BN}(P)$.

Notice that π_{010} and π_{110} are very similar to the (monadic/polyadic) asynchronous π-calculus [20, 4]; π_{101} relies on the communication paradigm adopted in LINDA [17]; π_{000} and π_{100} rely on the communication paradigm adopted in the (monadic/polyadic) Ambient Calculus [10]; π_{001} and π_{011} rely on the communication paradigm adopted in LCKLAIM and CKLAIM [13], respectively; finally, π_{111} relies on the communication paradigm adopted, e.g., in μKLAIM [13] or in semantic-π [11].

Remark 2.1. $\Pi_{a,m,p}$ can be easily ordered by language containment; in particular, $\pi_{\beta_1\beta_2\beta_3}$ can be seen as a sub-language of $\pi_{\beta'_1\beta'_2\beta'_3}$ if and only if, for every $i \in \{1,2,3\}$, it holds that $\beta_i \leq \beta'_i$. As an extremal example, consider π_{000} and π_{111}: monadic data are a particular case of polyadic data (all of length one); a shared dataspace can be modelled by letting all communications happen on the same global channel, say `ether`; finally, absence of pattern-matching can be obtained by only considering templates without actual fields.

Operational semantics The operational semantics of the calculi is given by means of a *labelled transition system* (LTS) describing the actions a process can perform to

evolve. Judgements take the form $P \xrightarrow{\alpha} P'$, meaning that P can become P' upon execution of α. *Labels* take the form

$$\alpha \;::=\; \tau \;\mid\; a?\widetilde{b} \;\mid\; (\nu\widetilde{c})a!\widetilde{b} \;\mid\; ?\widetilde{b} \;\mid\; (\nu\widetilde{c})!\widetilde{b}$$

Traditionally, τ denotes an internal computation; $a?\widetilde{b}$ and $(\nu\widetilde{c})a!\widetilde{b}$ denote the reception/sending of a sequence of names $\widetilde{b}$ along channel a; when channels are not present (namely, in $\pi_{\_0\_}$), $?\widetilde{b}$ and $(\nu\widetilde{c})!\widetilde{b}$ denote the withdrawal/emission of $\widetilde{b}$ from/in the shared dataspace. In $(\nu\widetilde{c})a!\widetilde{b}$ and $(\nu\widetilde{c})!\widetilde{b}$, some of the sent names, viz. $\widetilde{c}$ ($\subseteq \widetilde{b}$), are restricted. Notationally, $(\nu\widetilde{c})\text{-}!\widetilde{b}$ stands for either $(\nu\widetilde{c})a!\widetilde{b}$ or $(\nu\widetilde{c})!\widetilde{b}$; similarly, $\text{-}?\widetilde{b}$ stands for either $a?\widetilde{b}$ or $?\widetilde{b}$. As usual, $\text{BN}((\nu\widetilde{c})\text{-}!\widetilde{b}) \triangleq \widetilde{c}$; $\text{FN}(\alpha)$ and $\text{N}(\alpha)$ are defined accordingly.

The LTS provides some rules shared by all the calculi; the different semantics are obtained from the axioms for input/output actions. The common rules, reported below, are an easy adaptation of an early-style LTS for the π-calculus; thus, we do not comment them and refer the interested reader to [26].

$$\frac{P \xrightarrow{?\widetilde{b}} P' \qquad Q \xrightarrow{!\widetilde{b}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \frac{P \xrightarrow{a?\widetilde{b}} P' \qquad Q \xrightarrow{a!\widetilde{b}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad n \notin \text{N}(\alpha)}{(\nu n)P \xrightarrow{\alpha} (\nu n)P'} \qquad \frac{P \xrightarrow{(\nu\widetilde{c})\text{-}!\widetilde{b}} P' \qquad n \in \text{FN}(\widetilde{b}) \setminus \{\text{-}, \widetilde{c}\}}{(\nu n)P \xrightarrow{(\nu n,\widetilde{c})\text{-}!\widetilde{b}} P'}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad \text{BN}(\alpha) \cap \text{FN}(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \frac{P \equiv P_1 \xrightarrow{\alpha} P_2 \equiv P'}{P \xrightarrow{\alpha} P'}$$

The structural equivalence, $\equiv$, rearranges a process to let it evolve according to the rules of the LTS. Its defining axioms are the standard π-calculus' ones [26]:

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$

$$\textbf{if } n = n \textbf{ then } P \textbf{ else } Q \equiv P \qquad \textbf{if } n = m \textbf{ then } P \textbf{ else } Q \equiv Q \text{ if } n \neq m$$

$$P \equiv P' \text{ if } P =_\alpha P' \qquad (\nu n)\mathbf{0} \equiv \mathbf{0} \qquad (\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$$

$$P \mid (\nu n)Q \equiv (\nu n)(P \mid Q) \text{ if } n \notin \text{FN}(P) \qquad \textbf{rec}\, X.P \equiv P\{\textbf{rec}\, X.P/X\}$$

To define the semantics for the basic actions of the various calculi, we must specify when a template matches a datum. Intuitively, this happens whenever both have the same length and corresponding fields match (i.e., $\ulcorner n \urcorner$ matches n and x matches every name). This can be formalised via a partial function, called *pattern-matching* and written MATCH, that also returns a substitution σ; the latter will be applied to the process that performed the input to replace template formal fields with the corresponding

names of the datum retrieved. These intuitions are formalised by the following rules:

$$\textsc{Match}(\ ;\ ) = \epsilon \qquad \textsc{Match}(\ulcorner n \urcorner; n) = \epsilon \qquad \textsc{Match}(x; n) = \{n/x\}$$

$$\frac{\textsc{Match}(T; b) = \sigma_1 \qquad \textsc{Match}(\widetilde{T}; \widetilde{b}) = \sigma_2}{\textsc{Match}(T, \widetilde{T}\ ;\ b, \widetilde{b}) = \sigma_1 \circ \sigma_2}$$

where 'ϵ' denotes the empty substitution and '$\circ$' denotes substitution composition. Now, the operational rules for output/input actions in calculi π_{-0-} are

$$\langle \widetilde{b} \rangle \xrightarrow{!\widetilde{b}} \mathbf{0} \qquad (\widetilde{T}).P \xrightarrow{?\widetilde{b}} P\sigma \ \text{ if } \textsc{Match}(\widetilde{T}\ ;\ \widetilde{b}) = \sigma$$

and, similarly, the rules for calculi π_{-1-} are

$$\overline{a}\langle \widetilde{b} \rangle \xrightarrow{a!\widetilde{b}} \mathbf{0} \qquad a(\widetilde{T}).P \xrightarrow{a?\widetilde{b}} P\sigma \ \text{ if } \textsc{Match}(\widetilde{T}\ ;\ \widetilde{b}) = \sigma$$

Notation A substitution σ is a finite partial mapping of names for names; $P\sigma$ denotes the (capture avoiding) application of σ to P. As usual, we let $\Rightarrow$ stand for $\xrightarrow{\tau}{}^*$ (i.e., the reflexive and transitive closure of $\xrightarrow{\tau}$) and $\stackrel{\alpha}{\Longrightarrow}$ stand for $\Rightarrow\xrightarrow{\alpha}\Rightarrow$. We shall write $P \xrightarrow{\alpha}$ to mean that there exists a process P' such that $P \xrightarrow{\alpha} P'$; a similar notation is adopted for $P \Rightarrow$ and $P \stackrel{\alpha}{\Longrightarrow}$. Moreover, we let ϕ range over visible actions (i.e. labels different from τ) and ρ to range over (possibly empty) sequences of visible actions. Formally, $\rho ::= \varepsilon \mid \phi \cdot \rho$, where '$\varepsilon$' denotes the empty sequence of actions and '$\cdot$' represents concatenation; then, $N \stackrel{\varepsilon}{\Longrightarrow}$ is defined as $N \Rightarrow$ and $N \xrightarrow{\phi\cdot\rho}$ is defined as $N \stackrel{\phi}{\Longrightarrow}\stackrel{\rho}{\Longrightarrow}$.

We conclude the presentation of the languages with a Proposition collecting together some properties of the LTSs we have just defined, that will be useful in the sequel. The proof of these results easily follows from the definition of the LTSs.

Proposition 2.1. *The following facts hold:*

1. *if $P \in \pi_{--0}$ and $P \xrightarrow{-?\widetilde{b}}$, then $P \xrightarrow{-?\widetilde{c}}$ for every $\widetilde{c}$ of the same length as $\widetilde{b}$;*
2. *if $P \xrightarrow{\tau} P'$ then $P \equiv P_1 \mid P_2$ and $P' \equiv (\nu\widetilde{c})(P_1' \mid P_2')$, where*
   - *$P_1 \xrightarrow{?\widetilde{b}} P_1'$ and $P_2 \xrightarrow{(\nu\widetilde{c})!\widetilde{b}} P_2'$, whenever $P \in \pi_{-0-}$*
   - *$P_1 \xrightarrow{a?\widetilde{b}} P_1'$ and $P_2 \xrightarrow{(\nu\widetilde{c})a!\widetilde{b}} P_2'$, whenever $P \in \pi_{-1-}$*

   or vice versa.

3 Quality of an Encoding and Overview of our Results

We now study the relative expressive power of the calculi in $\Pi_{a,m,p}$ by trying to encode one in another. Formally, an *encoding* $[\![\cdot]\!]$ is a function mapping terms of the source language into terms of the target language. As already said, the relative expressive power of our calculi can be established by defining some criteria to evaluate the quality of the encodings or to prove impossibility results. We have informally discussed some key criteria in the Introduction; here, we define them formally and sum-up some of the arguments that justify them. Moreover, we also add some other, more natural and less debatable, requirements.

The main requirement, that we call *faithfulness*, is that the encoding must not change the semantics of a source term, i.e. it must preserve the observable behaviour of the term without introducing new behaviours. As very clearly discussed in [24], there are several ways to formalise this idea; we shall define it in the simplest possible way, by means of *barbs* and *divergence*.

Definition 3.1 (Barbs and Divergence). P *offers a* barb, *written* $P \Downarrow$, *iff* $P \xRightarrow{(\nu \widetilde{c})\overline{a}!\widetilde{b}}$. P diverges, *written* $P \Uparrow$, *iff* $P \xrightarrow{\tau}{}^{\omega}$.

The idea is to identify a basic observable behaviour (or *barb*) for the languages considered and require that the encoding preserves and reflects it (i.e., the encoding should maintain all the original barbs without introducing new ones). In the setting of an asynchronous language [2, 6], a barb is the possibility of emitting some datum.[1] Since barb preservation and reflection alone are too weak, it is also required that the computations of a process correspond to the computations of its encoding, and vice versa; this property is usually known as *operational correspondence*. Barb preservation and operational correspondence together yield *(weak) barbed bisimulation* [23, 2] that, however, is insensitive to divergence (i.e., it can equate a term with an infinite computation and a term with only finite computations). In our setting, it is clearly undesirable to have an encoding that turns a terminating term into a divergent one, since this would change the behaviour of the source term. So, we need a further requirement stating that also divergence must be preserved and reflected by the encoding.

Finally, a good encoding cannot depend on the particular names involved in the source process, since we are dealing with a family of name-passing calculi; we call this property *name invariance*. Furthermore, the encoding should not decrease the degree of parallelism in favour of centralised entities that control the behaviour of the encoded term: if we can find some process behaviour that cannot be implemented in the target language with the same degree of distribution as in the source one, then surely the former language will be "weaker" than the latter one. We express this last property as *homomorphism w.r.t.* '$|$'.

To sum up, we consider an encoding as a *'reasonable'* means to compare the expressive power of two languages if it enjoys all the properties discussed so far.

[1] We choose here a very weak form of barbs. This fact strengthens our impossibility results; on the other hand, our possibility results are not undermined by this choice, since they would also enjoy properties expressed in terms of more signifi cant barbs, such as those in [2, 6].

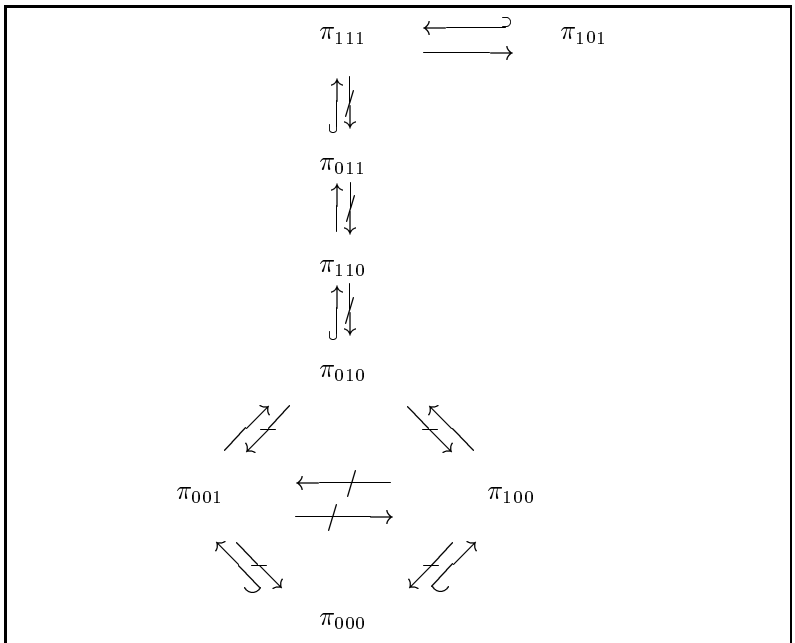

Table 1: Overview of the Results

Definition 3.2 (Reasonable Encoding). *An encoding* $[\![\cdot]\!]$ *is* reasonable *if it enjoys the following properties:*

1. (homomorphism w.r.t. '|'): $[\![P_1 | P_2]\!] \triangleq [\![P_1]\!] \mid [\![P_2]\!]$
2. (name invariance): $[\![P\sigma]\!] \triangleq [\![P]\!]\sigma$, *for every permutation of names* σ
3. (faithfulness): $P \Downarrow$ *iff* $[\![P]\!] \Downarrow$; $P \Uparrow$ *iff* $[\![P]\!] \Uparrow$
4. (operational correspondence):
   (a) *if* $P \xrightarrow{\tau} P'$ *then* $[\![P]\!] \stackrel{\tau}{\Longrightarrow} [\![P']\!]$
   (b) *if* $[\![P]\!] \xrightarrow{\tau} Q$ *then there exists a* P' *such that* $P \Rightarrow P'$ *and* $Q \Rightarrow [\![P']\!]$

The results of our paper are summarised in Table 1. It is worth noting that all the languages are Turing complete, as formally proved in Section 4.1. Moreover, notice that Definition 3.2(4).b is a weak form of correspondence; this makes our impossibility results stronger. However, for the encodability results, a better definition should keep into account every possible computation $[\![P]\!] \Rightarrow Q$. With this stronger property, the encodings provided in Sections 4.3 and 4.5 would not enjoy operational correspondence; we leave for future work the task of establishing whether encodings of π_{001} and π_{110} in π_{010} and π_{011} satisfying this stronger property exist or not.

4 Positive Results

In this Section we present the "positive" results; more precisely, we first show that all languages in $\Pi_{a,m,p}$ are Turing complete and then present the "$\rightarrow$" arrows of Table 1. For the latter task, we shall describe only the translation of the input and output actions; the remaining operators will be translated homomorphically (this trivially satisfies Definition 3.2(1)). Moreover, in what follows we are going to prove only that the encodings do not introduce divergence; preservation of divergence is a trivial consequence of Definition 3.2(4).a; Definition 3.2(2) and barb preservation/reflection will hold by construction of the encodings; Definition 3.2(4) can be routinely proved.

4.1 Turing Completeness

We now prove that π_{000} is Turing complete; by Remark 2.1, this implies that all languages in $\Pi_{a,m,p}$ are Turing complete. To this aim, we shall encode in π_{000} the language **L** from [6], that in *loc. cit.* is proved to be Turing complete.

Recall from [6] that the language **L** can be considered as the asyncronous version of CCS, without the relabelling operator. Formally, its syntax can be described as

$$P \ ::= \ \mathbf{0} \ \mid \ \langle a \rangle \ \mid \ P_1 \,|\, P_2 \ \mid \ (\nu a)P \ \mid \ X \ \mid \ \mathbf{rec}\, X.P \ \mid \ \Sigma_{i=1}^{n}(\ulcorner a_i \urcorner).P_i$$

The only new construct is $\Sigma_{i=1}^{n}(\ulcorner a_i \urcorner).P_i$, i.e. the input-guarded choice [2] among n (≥ 1) processes prefixed by an input action with an actual field. As usual, the operational semantics of **L** can be modelled by only adding rule

$$\Sigma_{i=1}^{n}(\ulcorner a_i \urcorner).P_i \ \xrightarrow{?a_i} \ P_i$$

and by stating that the choice is a monoidal operator, with $\mathbf{0}$ as identity (this requirement can be incorporated in the structural equivalence).

The encoding of **L** in π_{000} can be defined homomorphically for all the constructs but for input-guarded choice, that is encoded as follows:

$$\begin{aligned}[\![\, \Sigma_{i=1}^{n}(\ulcorner a_i \urcorner).P_i \,]\!] \ \triangleq \ \ & \mathbf{rec}\, X.(x).\mathbf{if}\ x = a_1\ \mathbf{then}\ [\![\, P_1 \,]\!]\ \mathbf{else} \\ & \qquad \mathbf{if}\ x = a_2\ \mathbf{then}\ [\![\, P_2 \,]\!]\ \mathbf{else} \\ & \qquad\quad \cdots \\ & \qquad\quad \mathbf{if}\ x = a_n\ \mathbf{then}\ [\![\, P_n \,]\!]\ \mathbf{else}\ \langle x \rangle \mid X\end{aligned}$$

Such an encoding is *not* reasonable because it introduces divergence. Consider, e.g., the **L**-process

$$\langle a \rangle \ \mid \ (\ulcorner b \urcorner).P$$

(that, incidentally, is also a π_{001}-process). For $a \neq b$ the process is blocked; nevertheless, its encoding

$$\langle a \rangle \quad \mid \quad \mathbf{rec}\, X.(x).\mathbf{if}\ x = b\ \mathbf{then}\ [\![\, P \,]\!]\ \mathbf{else}\ \langle x \rangle \mid X$$

[2] Actually, [6] presents a more general kind of choice, viz. $P_1 + P_2$; however, to prove Turing completeness, only input-guarded choice is exploited. For this reason, we consider here this simpler construct.

is a clearly divergent π_{000}-process. However, the fact that the encoding introduces divergence is irrelevant to study Turing completeness; to this aim, the crucial requirements are operational correspondence and barb preservation, and the encoding we have just presented enjoys them (the proof of this fact is simple and left to the interested reader).

4.2 An Encoding of π_{111} in π_{101}

The only feature of π_{111} not present in π_{101} is the possibility of specifying the name of a channel where the exchange happens. However, thanks to pattern-matching, this feature can be very easily encoded in π_{101}: it suffices to impose that the first name of every datum represents the name of the channel where the interaction is scheduled and that every template starts with the corresponding actual field. This discipline is rendered by the following encoding:

$$\begin{aligned} [\![\, \overline{a}\langle \widetilde{b} \rangle \,]\!] &\triangleq \langle a, \widetilde{b} \rangle \\ [\![\, a(\widetilde{T}).P \,]\!] &\triangleq (\ulcorner a \urcorner, \widetilde{T}).[\![\, P \,]\!] \end{aligned}$$

Proposition 4.1. *The encoding $[\![\, \cdot \,]\!] : \pi_{111} \longrightarrow \pi_{101}$ is reasonable.*

Proof. Definition 3.2(2) holds by construction. Definition 3.2(4).b can be proved as a stronger claim: if $[\![\, P \,]\!] \xrightarrow{\tau} Q$, then $Q \equiv [\![\, P' \,]\!]$ and $P \xrightarrow{\tau} P'$ (this result, like Definition 3.2(4).a, is proved by an easy induction over the shortest inference for $\xrightarrow{\tau}$). Definition 3.2(3) holds easily; just notice that the stronger formulation of operational correspondence mentioned above implies that the encoding cannot introduce divergence. □

4.3 An Encoding of π_{001} in π_{010}

We now have to translate the monadic pattern-matching of π_{001} into the channel-based exchanges of π_{010}. This would have been an easy task, if only actual fields occurred in templates: indeed, $\langle b \rangle$ would have been translated in $\overline{b}\langle b \rangle$ and, correspondingly, $(\ulcorner b \urcorner).P$ would have been translated in $b(y).[\![\, P \,]\!]$, for y fresh. This encoding, however, does not work well when trying to translate $(x).P$.

Thus, $\langle b \rangle$ in π_{001} should correspond to two outputs in π_{010}: one over b, to mimic name matching as described above, and one over a fresh and reserved channel `ether`, to enable inputs with formal fields. Symmetrically, an input action is translated in two successive inputs: the first one from `ether` and the second one from the received value, if we are translating an input with a formal field, and vice versa, otherwise. For example, $\langle b \rangle \mid \langle c \rangle \mid (\ulcorner \tilde{c} \urcorner).P$ is translated to $\overline{b}\langle b \rangle \mid \overline{\texttt{ether}}\langle b \rangle \mid \overline{c}\langle c \rangle \mid \overline{\texttt{ether}}\langle c \rangle \mid c(y).\texttt{ether}(z).[\![\, P \,]\!]$. We believe that this encoding is reasonable, but proving that it does not introduce divergence is hard because of the possible interferences between parallel components (e.g., the above example could evolve in $\overline{b}\langle b \rangle \mid \overline{\texttt{ether}}\langle c \rangle \mid [\![\, P \,]\!]$, by performing a communication between $c(y)$ and $\overline{c}\langle c \rangle$ and between $\texttt{ether}(z)$ and $\overline{\texttt{ether}}\langle b \rangle$).

The problem is that the two outputs in the translation of $\langle b \rangle$ are totally unrelated. This can be fixed by associating every output with a restricted name and by using such a name to reduce the effects of interferences. Formally,

$$\begin{aligned}
[\![\langle b \rangle]\!] &\triangleq (\nu n)(\overline{\texttt{ether}}\langle n \rangle \mid \overline{b}\langle n \rangle \mid \overline{n}\langle b \rangle) \\
[\![(x).P]\!] &\triangleq \texttt{ether}(y).y(x).x(z).\mathbf{if}\ y = z\ \mathbf{then}\ [\![P]\!] \\
[\![(\ulcorner b \urcorner).P]\!] &\triangleq b(y).y(y').\texttt{ether}(z).\mathbf{if}\ y = z\ \mathbf{then}\ [\![P]\!]
\end{aligned}$$

for n, y, y' and z fresh names. Clearly, this solution does not rule out interferences; it simply blocks interfering processes. This suffices to make the proof of reasonableness easier; to this aim, the key result is the following Lemma.

Lemma 4.2. *Let κ be the number of top-level outputs in P. If $[\![P]\!] \xrightarrow{\tau}{}^{3\kappa+1} Q$, then there exists a P' such that $P \xrightarrow{\tau} P'$ and $[\![P]\!] \xrightarrow{\tau}{}^{3} [\![P']\!] \Rightarrow Q$.*

Proof. First notice that we need three τ-steps to consume the encoding of some $\langle b \rangle$. Thus, since $[\![P]\!]$ performs $3\kappa + 1$ τ-steps, this means that at least the encoding of one output has been fully consumed and that a test put forward by an $\mathbf{if\text{-}then}$ has been successfully passed. By definition of the encoding, this happens if and only if at least the encoding of an output has been consumed without interferences in the computation leading $[\![P]\!]$ to Q;[3] let $\langle b \rangle$ be the first of such outputs. By construction, $\langle b \rangle$ is a top-level output in P; thus, $P \equiv (\nu \widetilde{n})(\langle b \rangle \mid P_1 \mid P_2)$, where P_1 is the parallel component whose encoding consumes $[\![\langle b \rangle]\!]$. Recall that $[\![\cdot]\!]$ is a homomorphism w.r.t. restrictions and parallel compositions; moreover, it is trivial to prove that it maps structurally equivalent π_{001}-processes into structurally equivalent π_{010}-processes. Thus, $[\![P]\!] \equiv (\nu \widetilde{n})([\![\langle b \rangle]\!] \mid [\![P_1]\!] \mid [\![P_2]\!])$; moreover, if $[\![P]\!] \Rightarrow Q$, then $Q \equiv (\nu \widetilde{n})R$ and $[\![\langle b \rangle]\!] \mid [\![P_1]\!] \mid [\![P_2]\!] \Rightarrow R$. The latter reduction holds if and only if $[\![P_1]\!] \mid [\![P_2]\!] \Rightarrow [\![P_1]\!] \mid Q_1 \xrightarrow{\alpha_1} P_1' \mid Q_1 \Rightarrow P_1' \mid Q_2 \xrightarrow{\alpha_2} P_1'' \mid Q_2 \Rightarrow P_1'' \mid Q_3 \xrightarrow{\alpha_3} P_1''' \mid Q_3 \Rightarrow R$, for $[\![P_1]\!] \xrightarrow{\alpha_1} P_1' \xrightarrow{\alpha_2} P_1'' \xrightarrow{\alpha_3} P_1'''$ and $[\![P_2]\!] \Rightarrow Q_1 \Rightarrow Q_2 \Rightarrow Q_3$. This holds because P_1 is the parallel component whose encoding consumes without interferences $[\![\langle b \rangle]\!]$ and $\langle b \rangle$ is the first output whose encoding has been consumed without interferences. Then, $[\![P_1]\!] \mid [\![P_2]\!] \xrightarrow{\alpha_1} \xrightarrow{\alpha_2} \xrightarrow{\alpha_3} P_1''' \mid [\![P_2]\!] \Rightarrow R$. Now,

(a) if $P_1 \triangleq (x).P_3$ then $\alpha_1 \triangleq \texttt{ether}?n$, $\alpha_2 \triangleq n?b$, $\alpha_3 \triangleq b?n$ and $P_1''' \equiv [\![P_3\{^b/_x\}]\!]$

(b) if $P_1 \triangleq (\ulcorner b \urcorner).P_3$ then $\alpha_1 \triangleq b?n$, $\alpha_2 \triangleq n?b$, $\alpha_3 \triangleq \texttt{ether}?n$ and $P_1''' \equiv [\![P_3]\!]$.

In both cases, $[\![P]\!] \xrightarrow{\tau}{}^{3} (\nu \widetilde{n})(P_1''' \mid [\![P_2]\!]) \Rightarrow Q$; we conclude by letting $P' \triangleq (\nu \widetilde{n})(P_3\{^b/_x\} \mid P_2)$ in case (a) and $P' \triangleq (\nu \widetilde{n})(P_3 \mid P_2)$ in case (b). □

Proposition 4.3. *The encoding $[\![\cdot]\!] : \pi_{001} \longrightarrow \pi_{010}$ is reasonable.*

Proof. We only prove that $[\![\cdot]\!]$ does not introduce divergence; the other requirements are simple. Assume that $[\![P]\!] \Uparrow$, i.e. there exists an infinite computation $[\![P]\!] \triangleq Q_0 \xrightarrow{\tau} Q_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} Q_{3\kappa+1} \xrightarrow{\tau} \cdots$. By Lemma 4.2, there exists a P'

[3] Notice that, without the restricted name associated to the encoding of an output and without the corresponding $\mathbf{if\text{-}then}$ in the encoding of an input, this crucial property would not hold.

such that $[\![P]\!] \xrightarrow{\tau}{}^{3} [\![P']\!] \Rightarrow Q_{3\kappa+1}$ and $P \xrightarrow{\tau} P'$. But $[\![P']\!]$ is still divergent; indeed, $[\![P']\!] \Rightarrow Q_{3\kappa+1} \xrightarrow{\tau} \cdots$. By iterating this reasoning, we can build up a divergent computation for P, i.e. $P \xrightarrow{\tau} P' \xrightarrow{\tau} \cdots$; hence, $[\![\cdot]\!]$ does not introduce divergence. □

4.4 An Encoding of π_{100} in π_{010}

The only feature of π_{100} is that it can check the arity of a datum before retrieving it (see the definition of function MATCH). This, however, can be mimicked by the channel-based communication of π_{010}. Indeed, we assume a (reserved) channel for every possible arity: a datum of arity k will be represented as an output over channel $\mathtt{k}$; an input of arity k will be represented as an input from $\mathtt{k}$; a communication over $\mathtt{k}$ in π_{010} can happen if and only if pattern-matching succeeds in π_{100}; finally, the exchanged datum is a restricted name that will be used in the actual data exchange.

The encoding assumes that $\mathtt{0}, \mathtt{1}, \ldots, \mathtt{k}, \ldots$ are fresh and reserved names; then

$$[\![\langle b_1, \cdots, b_k \rangle]\!] \triangleq (\nu n)(\overline{\mathtt{k}}\langle n \rangle \mid n(y).(\overline{y}\langle b_1 \rangle \mid n(y).(\overline{y}\langle b_2 \rangle \mid n(y).(\cdots n(y).\overline{y}\langle b_k \rangle) \cdots)))$$

$$[\![(x_1, \cdots, x_k).P]\!] \triangleq \mathtt{k}(x).(\nu m)(\overline{x}\langle m \rangle \mid m(x_1).(\overline{x}\langle m \rangle \mid m(x_2).(\cdots (\overline{x}\langle m \rangle \mid m(x_k).[\![P]\!]) \cdots)))$$

for n, m, x and y fresh names. The datum emitted over $\mathtt{k}$ (viz. n) is used as a "synchroniser", to keep the order of the transmitted data and force the right name-to-variable association. The actual exchange takes place over a restricted channel created by the receiver (viz. m) and transmitted along n as an ack to the sender.

Similarly to Proposition 4.3, reasonableness of this encoding can be easily proved, once we prove the following Lemma. Moreover, notice that for this encoding the stronger version of Definition 3.2(4).b holds: if $[\![P]\!] \Rightarrow Q$, then there is a P' such that $P \Rightarrow P'$ and $Q \Rightarrow [\![P']\!]$.

Lemma 4.4. *Let κ and λ be the number and the maximum arity of top-level outputs in P, respectively. If $[\![P]\!] \xrightarrow{\tau}{}^{\kappa(2\lambda+1)} Q$, then there exists a P' such that $P \xrightarrow{\tau} P'$ and $[\![P]\!] \xrightarrow{\tau}{}^{2h+1} [\![P']\!] \Rightarrow Q$, for $0 \leq h \leq \lambda$.*

Proof. We work like in the proof of Lemma 4.2. Now, the encoding of a h-ary output is consumed (without the risk of interferences) in $2h+1$ τ-steps. Thus, upon execution of $\kappa(2\lambda+1)$ τ-steps, at least the encoding of an output has been fully consumed without interferences; let $\langle\widetilde{b}\rangle$ be the first of such outputs and h be its arity. We then proceed like before. □

Proposition 4.5. *The encoding $[\![\cdot]\!] : \pi_{100} \longrightarrow \pi_{010}$ is reasonable.*

4.5 An Encoding of π_{110} in π_{011}

In π_{110}, a communication succeeds if (and only if) a datum of a proper length is present over the channel specified by the inputting process. So, two kinds of information are atomically verified: the length of the message and the channel where it should be

transmitted. This can be mimicked in π_{011} by having one fresh and reserved name for every length (say, $\mathtt{0}, \mathtt{1}, \ldots, \mathtt{k}, \ldots$); a k-ary input from a is then translated into a process starting with $a(\ulcorner \mathtt{k} \urcorner)$ and, correspondingly, a k-ary output on a is translated into a process offering $\mathtt{k}$ at a. Once this communication took place, we are sure that a k-ary datum is available on a; we then proceed similarly to Section 4.4 for the actual data exchange: a new channel n is made available on a, to maintain the order of messages, while a new channel m is sent back on n to transmit the datum name by name.

However, we need to enhance the encoding of Section 4.4 to avoid interferences due to the fact that the existence of a k-ary output and the acquisition of the new name for the actual exchange are not atomic here. Indeed, the translation of $\overline{a}\langle b_1, b_2\rangle \mid \overline{a}\langle c_1, c_2, c_3\rangle \mid a(x_1, x_2).P \mid a(x_1, x_2, x_3).Q$ can originate interferences that can lead to divergence. Thus, like in Section 4.3, we shall verify at the end of the data exchange the consistency of the exchange, i.e. that a k-ary data has really been retrieved. To this aim, let $\mathtt{end}$ be another fresh and reserved name; then

$$\begin{aligned}
[\![\, \overline{a}\langle b_1, \cdots, b_k\rangle \,]\!] &\triangleq \overline{a}\langle \mathtt{k}\rangle \mid (\nu n)(\overline{a}\langle n\rangle \mid n(y).(\overline{y}\langle b_1\rangle \mid \cdots\ n(y).(\overline{y}\langle b_k\rangle \mid \\
&\qquad\qquad n(y).\overline{y}\langle \mathtt{end}\rangle)\cdots)) \\
[\![\, a(x_1, \cdots, x_k).P \,]\!] &\triangleq a(\ulcorner \mathtt{k} \urcorner).a(x).(\nu m)(\overline{x}\langle m\rangle \mid m(x_1).(\cdots \mid (\overline{x}\langle m\rangle \mid m(x_k). \\
&\qquad\qquad (\overline{x}\langle m\rangle \mid m(\ulcorner \mathtt{end} \urcorner).[\![P]\!])\cdots)))
\end{aligned}$$

for n, m, x and y fresh names. Reasonableness of this encoding can be proved like in Proposition 4.3, as a consequence of the following Lemma.

Lemma 4.6. *Let κ and λ be the number and the maximum arity of top-level outputs in P, respectively. If $[\![P]\!] \xrightarrow{\tau}{}^{\kappa(2\lambda+3)+1} Q$, then there exists a P' such that $P \xrightarrow{\tau} P'$ and $[\![P]\!] \xrightarrow{\tau}{}^{2h+4} [\![P']\!] \Rightarrow Q$, for $0 \leq h \leq \lambda$.*

Proof. The proof is similar to the proof of Lemma 4.4. Now, the encoding of a h-ary output is consumed in $2h+4$ τ-steps and, upon execution of $\kappa(2\lambda+3)+1$ τ-steps, at least the encoding of an output has been fully consumed without interferences; let $\overline{a}\langle \widetilde{b}\rangle$ be the first of such outputs and $h = |\widetilde{b}|$. Then, $P \equiv (\nu \widetilde{n})(\overline{a}\langle \widetilde{b}\rangle \mid P_1 \mid P_2)$, $P_1 \triangleq (\widetilde{x}).P_3$, $|\widetilde{x}| = h$ (otherwise action $m(\ulcorner \mathtt{end} \urcorner)$ in the encoding of P_1 would not succeed), $Q \equiv (\nu \widetilde{n})R$ and $[\![\overline{a}\langle \widetilde{b}\rangle]\!] \mid [\![P_1]\!] \mid [\![P_2]\!] \Rightarrow R$. Again, the latter reduction holds if and only if $[\![P_1]\!] \mid [\![P_2]\!] \xrightarrow{a?\mathtt{k}} \xrightarrow{a?n} \xrightarrow{(\nu m)n!m} \xrightarrow{m?b_1} \cdots \xrightarrow{n!m} \xrightarrow{m?b_h} \xrightarrow{n!m} \xrightarrow{m?\mathtt{end}}$ $[\![P_3\{\widetilde{b}/\widetilde{x}\}]\!] \mid [\![P_2]\!] \Rightarrow R$. Then, $[\![P]\!] \xrightarrow{\tau}{}^{2h+4} (\nu \widetilde{n})([\![P_3\{\widetilde{b}/\widetilde{x}\}]\!] \mid [\![P_2]\!]) \Rightarrow Q$; we conclude by letting $P' \triangleq (\nu \widetilde{n})(P_3\{\widetilde{b}/\widetilde{x}\} \mid P_2)$. □

Proposition 4.7. *The encoding $[\![\cdot]\!] : \pi_{110} \longrightarrow \pi_{011}$ is reasonable.*

5 Impossibility Results

We now consider the impossibility results, i.e. the "$\not\rightarrow$" arrows of Table 1, that are the main technical contribution of this paper. They are all proved by contradiction: we assume that a reasonable encoding exists and show that it introduces divergence. Often, the contradiction is obtained by exhibiting a process that cannot reduce but

whose encoding reduces. This fact, together with operational correspondence, implies that the encoding introduces divergence, as stated by the following simple result.

Proposition 5.1. *Let $[\![\cdot]\!]$ be an operationally corresponding encoding. If there exists a process P such that $P \stackrel{\tau}{\not\longrightarrow}$ but $[\![P]\!] \stackrel{\tau}{\longrightarrow}$, then $[\![\cdot]\!]$ introduces divergence.*

Proof. The fact that $[\![P]\!] \stackrel{\tau}{\longrightarrow} Q$ implies, by operational correspondence, that $P \Rightarrow P'$, for some P' such that $Q \Rightarrow [\![P']\!]$. But the only P' such that $P \Rightarrow P'$ is P itself; thus, $[\![P]\!] \stackrel{\tau}{\longrightarrow}^{+} [\![P]\!]$, i.e. $[\![P]\!]$ diverges. □

Theorem 5.2. *There exists no reasonable encoding of π_{011} in π_{110}.*

Proof. Assume that $[\![\cdot]\!]$ is reasonable and consider the process $a(\ulcorner b \urcorner) \mid \overline{a}\langle b\rangle$, for $a \neq b$, that evolves in $\mathbf{0}$. By operational correspondence, $[\![a(\ulcorner b \urcorner) \mid \overline{a}\langle b\rangle]\!] \Rightarrow [\![\mathbf{0}]\!]$; moreover, by faithfulness, $[\![a(\ulcorner b \urcorner)]\!] \not\Downarrow$, $[\![\overline{a}\langle b\rangle]\!] \Downarrow$ and $[\![\mathbf{0}]\!] \not\Downarrow$. Thus, the barb of $[\![\overline{a}\langle b\rangle]\!]$ must be consumed in the computation leading $[\![a(\ulcorner b \urcorner) \mid \overline{a}\langle b\rangle]\!]$ to $[\![\mathbf{0}]\!]$.

Now, notice that $[\![\overline{a}\langle b\rangle]\!]$ cannot perform a τ-step otherwise, by Proposition 5.1, $[\![\cdot]\!]$ would introduce divergence. This fact, together with $[\![a(\ulcorner b \urcorner) \mid \overline{a}\langle b\rangle]\!] \triangleq [\![a(\ulcorner b \urcorner)]\!] \mid [\![\overline{a}\langle b\rangle]\!]$, implies that $[\![a(\ulcorner b \urcorner)]\!]$ consumed the barb offered by $[\![\overline{a}\langle b\rangle]\!]$. Thus, it must be that $[\![a(\ulcorner b \urcorner)]\!] \stackrel{n?\widetilde{c}}{\longrightarrow}$, for some n and $\widetilde{c}$ such that $[\![\overline{a}\langle b\rangle]\!] \stackrel{(\nu \widetilde{c'})n!\widetilde{c}}{\longrightarrow}$; by Proposition 2.1(1), this fact implies that $[\![a(\ulcorner b \urcorner)]\!] \stackrel{n?\widetilde{d}}{\longrightarrow}$, for every $\widetilde{d}$ of the same arity as $\widetilde{c}$, i.e. $|\widetilde{d}| = |\widetilde{c}|$.

If $n \neq b$, then pick up $e \notin \{a, b, n\}$ and the permutation of names swapping b and e; by name invariance, it holds that $[\![\overline{a}\langle e\rangle]\!] \stackrel{(\nu \widetilde{f'})n!\widetilde{f}}{\longrightarrow}$, where $\widetilde{f}$ and $\widetilde{f'}$ are the renaming of $\widetilde{c}$ and $\widetilde{c'}$. In particular, $|\widetilde{c}| = |\widetilde{f}|$. Then, $[\![a(\ulcorner b \urcorner) \mid \overline{a}\langle e\rangle]\!] \triangleq [\![a(\ulcorner b \urcorner)]\!] \mid [\![\overline{a}\langle e\rangle]\!] \stackrel{\tau}{\longrightarrow}$, while $a(\ulcorner b \urcorner) \mid \overline{a}\langle e\rangle \stackrel{\tau}{\not\longrightarrow}$. By Proposition 5.1, $[\![\cdot]\!]$ is not reasonable, as it introduces divergence; contradiction.

If $n = b$, then pick up $e \notin \{a, b\}$, the permutation of names swapping a and e, and work like before, with process $[\![a(\ulcorner b \urcorner) \mid \overline{e}\langle b\rangle]\!]$. □

Corollary 5.3. *There exists no reasonable encoding of π_{001} in π_{100}.*

Proof. Trivial consequence of Theorem 5.2, since π_{001} and π_{100} can be seen as the subcalculi of π_{011} and π_{110} where all the communications happen on the same (unique and global) channel. □

Theorem 5.4. *There exists no reasonable encoding of π_{010} in π_{100}.*

Proof. The proof is similar to that of Theorem 5.2. We start with process $\overline{a}\langle b\rangle \mid a(x)$, for $a \neq b$; it holds that $[\![\overline{a}\langle b\rangle]\!] \stackrel{(\nu \widetilde{c'})!\widetilde{c}}{\longrightarrow}$ and $[\![a(x)]\!] \stackrel{?\widetilde{c}}{\longrightarrow}$, for some $\widetilde{c}$. By name invariance, $[\![\overline{b}\langle a\rangle]\!] \stackrel{(\nu \widetilde{d'})!\widetilde{d}}{\longrightarrow}$, where $\widetilde{d}$ and $\widetilde{d'}$ are obtained by swapping a and b in $\widetilde{c}$ and $\widetilde{c'}$; thus, $|\widetilde{d}| = |\widetilde{c}|$. Now, $[\![a(x)]\!] \stackrel{?\widetilde{d}}{\longrightarrow}$ and $[\![\overline{b}\langle a\rangle \mid a(x)]\!] \stackrel{\tau}{\longrightarrow}$, while $\overline{b}\langle a\rangle \mid a(x) \stackrel{\tau}{\not\longrightarrow}$; this suffices to conclude. □

Theorem 5.5. *There exists no reasonable encoding of π_{110} in π_{010}.*

Proof. Similarly to the proof of Theorem 5.2, consider the process $a(x,y) \mid \overline{a}\langle b,c\rangle$; again, $[\![\, \overline{a}\langle b,c\rangle \,]\!] \xrightarrow{(\nu \widetilde{d'})n!d}$ and $[\![\, a(x,y) \,]\!] \xrightarrow{n?d}$, for some d and $\widetilde{d'}$. If $n \neq a$, choose $e \neq a$; by name invariance, $[\![\, \overline{e}\langle b,c\rangle \,]\!] \xrightarrow{(\nu \widetilde{f'})n!f}$ and $[\![\, a(x,y) \mid \overline{e}\langle b,c\rangle \,]\!] \xrightarrow{\tau}$, while $a(x,y) \mid \overline{e}\langle b,c\rangle \not\xrightarrow{\tau}$. If $n = a$, consider $a(x,y,z) \mid \overline{a}\langle b,c,c\rangle$; like before, $[\![\, \overline{a}\langle b,c,c\rangle \,]\!] \xrightarrow{(\nu \widetilde{e'})m!e}$ and $[\![\, a(x,y,z) \,]\!] \xrightarrow{m?e}$. Then, if $m = a$, we have that $[\![\, a(x,y) \mid \overline{a}\langle b,c,c\rangle \,]\!] \xrightarrow{\tau}$; otherwise, choose $f \neq a$ and conclude that $[\![\, a(x,y,z) \mid \overline{f}\langle b,c,c\rangle \,]\!] \xrightarrow{\tau}$. □

Corollary 5.6. *There exist no reasonable encodings of π_{001} and π_{100} in π_{000}.*

Proof. Easily derivable from Corollary 5.2 and Theorem 5.5, respectively. □

Theorem 5.7. *There exists no reasonable encoding of π_{111} in π_{011}.*

Proof. Consider the process $\overline{a}\langle b,c\rangle \mid a(\ulcorner b \urcorner, \ulcorner c \urcorner)$, for a, b and c pairwise distinct. Like in Theorem 5.2, we have that $[\![\, a(\ulcorner b \urcorner, \ulcorner c \urcorner) \,]\!] \xrightarrow{n?m}$ and $[\![\, \overline{a}\langle b,c\rangle \,]\!] \xrightarrow{(\nu \widetilde{m})n!m}$. If the input of $[\![\, a(\ulcorner b \urcorner, \ulcorner c \urcorner) \,]\!]$ has been generated by relying on a template formal field, then $[\![\, a(\ulcorner b \urcorner, \ulcorner c \urcorner) \,]\!] \xrightarrow{n?l}$, for every l; by Proposition 5.1, this would suffice to build up a divergent computation for $[\![\, a(\ulcorner b \urcorner, \ulcorner c \urcorner) \mid \overline{a}\langle b,d\rangle \,]\!]$, for every $d \neq c$. Otherwise, the input of $[\![\, a(\ulcorner b \urcorner, \ulcorner c \urcorner) \,]\!]$ relies on an actual field; we then consider the following possibilities for n and m:

1. $c \notin \{n,m\}$: let $d \neq c$ and consider the permutation that swaps c and d; then, $[\![\, \overline{a}\langle b,d\rangle \,]\!] \xrightarrow{(\nu \widetilde{m})n!m}$ and $[\![\, \overline{a}\langle b,d\rangle \mid a(\ulcorner b \urcorner, \ulcorner c \urcorner) \,]\!] \xrightarrow{\tau}$.
2. $n = c$, $m \neq b$: let $d \neq b$ and consider the permutation that swaps b and d; like before, $[\![\, \overline{a}\langle d,c\rangle \mid a(\ulcorner b \urcorner, \ulcorner c \urcorner) \,]\!] \xrightarrow{\tau}$.
3. $n = c$, $m = b$: let $d \neq a$ and consider the permutation that swaps a and d; then, $[\![\, \overline{d}\langle b,c\rangle \mid a(\ulcorner b \urcorner, \ulcorner c \urcorner) \,]\!] \xrightarrow{\tau}$.
4. $m = c$, $n \neq b$: like case 2.
5. $m = c$, $n = b$: like case 3. □

Theorem 5.8. *There exists no reasonable encoding of π_{010} in π_{001}.*

Proof. By contradiction, assume that there exists a reasonable encoding $[\![\, \cdot \,]\!]$. Let a, b, c and d be pairwise distinct names; let Ω denote a divergent process and define $P \triangleq \mathbf{if}\ x = d\ \mathbf{then}\ \Omega$. Faithfulness implies that $[\![\, \mathbf{if}\ d = d\ \mathbf{then}\ \Omega \,]\!]$ diverges, $[\![\, a(x).P \,]\!]$ cannot offer data and $[\![\, \overline{a}\langle b\rangle \,]\!]$ must offer some datum. Moreover, because of Proposition 5.1, $[\![\, a(x).P \,]\!]$ and $[\![\, \overline{a}\langle b\rangle \,]\!]$ cannot perform τ-steps in isolation; however, because of operational correspondence, when put in parallel they must perform at least one τ-step to become $[\![\, P\{{}^{b}/_{x}\} \,]\!]$. If the input performed by $[\![\, a(x).P \,]\!]$ relies on a formal field, then we can obtain a divergent computation from $[\![\, a(x).P \mid \overline{c}\langle b\rangle \,]\!]$. So, it must be that $[\![\, a(x).P \,]\!]$ starts with an input relying on an actual template field $\ulcorner a \urcorner$ (it must be a otherwise, by name invariance, $[\![\, a(x).P \mid \overline{c}\langle b\rangle \,]\!]$ would diverge).

Now, by exploiting Proposition 2.1(2) and the definition of the LTS, we have that $[\![a(x).P]\!] \mid [\![\overline{a}\langle b\rangle]\!] \Rightarrow [\![P\{^b/_x\}]\!]$ if and only if $[\![a(x).P]\!] \stackrel{\rho}{\Longrightarrow} R$, $[\![\overline{a}\langle b\rangle]\!] \stackrel{\overline{\rho}}{\Longrightarrow} R'$ and $R \mid R' \equiv [\![P\{^b/_x\}]\!]$. It must be that $\rho \triangleq ?a \cdot \rho_1 \cdot ?b \cdot \rho_2$, for b not occurring in ρ_1 and $?b$ generated by an input action with a formal template field; moreover, $[\![a(x).P]\!] \stackrel{?a\cdot\rho_1}{\Longrightarrow} R_1 \stackrel{?b}{\longrightarrow} R_2 \stackrel{\rho_2}{\Longrightarrow} R$ and $[\![\overline{a}\langle b\rangle]\!] \stackrel{!a\cdot\overline{\rho_1}}{\Longrightarrow} R_3 \stackrel{!b}{\longrightarrow} R_4 \stackrel{\overline{\rho_2}}{\Longrightarrow} R'$. In particular, $\rho_2 \triangleq \Box_1 n_1 \cdot \ldots \cdot \Box_k n_k$, for $\Box_i \in \{?,!\}$, and $\overline{\rho_2} \triangleq \Diamond_1 n_1 \cdot \ldots \cdot \Diamond_k n_k$, for $\Diamond_i \in \{?,!\} - \{\Box_i\}$.

Let σ be the permutation that swaps a and c and b and d; by name invariance, $[\![c(x).P]\!] \stackrel{?c\cdot\rho_1'}{\Longrightarrow} R_1\sigma \stackrel{?d}{\longrightarrow} R_2\sigma \stackrel{\rho_2'}{\Longrightarrow} R\sigma$ and $[\![\overline{c}\langle d\rangle]\!] \stackrel{!c\cdot\overline{\rho_1'}}{\Longrightarrow} R_3\sigma \stackrel{!d}{\longrightarrow} R_4\sigma \stackrel{\overline{\rho_2'}}{\Longrightarrow} R'\sigma$, for $\rho_1' = \rho_1\sigma$ and $\rho_2' = \rho_2\sigma$. More precisely, $\rho_2' \triangleq \Box_1 n_1' \cdot \ldots \cdot \Box_k n_k'$ and $\overline{\rho_2'} \triangleq \Diamond_1 n_1' \cdot \ldots \cdot \Diamond_k n_k'$, for $n_i' \triangleq \sigma(n_i)$.

Now, consider $Q \triangleq a(x).P \mid \overline{a}\langle b\rangle \mid \overline{c}\langle d\rangle \mid c(x).P'$, where $P' \triangleq \mathbf{if}\ x = b\ \mathbf{then}\ \Omega$; trivially, $Q \not\Downarrow$ while, as we shall see, $[\![Q]\!] \Uparrow$. This yields the desired contradiction. Consider

$$[\![Q]\!] \Rightarrow R_1 \mid R_3 \mid R_1\sigma \mid R_3\sigma \rightarrow\!\!\rightarrow R_2\{^d/_b\} \mid R_4 \mid (R_2\sigma)\{^b/_d\} \mid R_4\sigma$$

where R_1 received d in place of b and $R_1\sigma$ received b in place of d (this is possible since these inputs do not rely on actual fields). Now, $R_2\{^d/_b\} \stackrel{\Box_1 m_1\cdot\ldots\cdot\Box_k m_k}{\Longrightarrow}$, where

$$m_k \triangleq \begin{cases} d & \text{if } n_i = b \\ n_i & \text{otherwise} \end{cases}$$

and $(R_2\sigma)\{^b/_d\} \stackrel{\Box_1 m_1'\cdot\ldots\cdot\Box_k m_k'}{\Longrightarrow}$, where

$$m_k' \triangleq \begin{cases} b & \text{if } n_i' = d \\ n_i' & \text{otherwise} \end{cases}$$

Finally, consider the computation

$$R_2\{^d/_b\} \mid R_4 \mid (R_2\sigma)\{^b/_d\} \mid R_4\sigma \Rightarrow R\{^d/_b\} \mid R'\{^d/_b\} \mid (R\sigma)\{^b/_d\} \mid (R'\sigma)\{^b/_d\}$$

obtained by performing a communication

- between $\Box_i m_i$ and $\Diamond_i n_i$ and between $\Box_i m_i'$ and $\Diamond_i n_i'$, if $n_i \neq b$, or
- between $\Box_i m_i$ and $\Diamond_i n_i'$ and between $\Box_i m_i'$ and $\Diamond_i n_i$, otherwise.

Now, $R\{^d/_b\} \mid R'\{^d/_b\} \triangleq (R \mid R')\{^d/_b\} \equiv [\![P\{^b/_x\}]\!]\{^d/_b\} \triangleq [\![\mathbf{if}\ d = d\ \mathbf{then}\ \Omega]\!]$, that is a divergent process. □

Theorem 5.9. *There exists no reasonable encoding of π_{100} in π_{001}.*

Proof. The proof is similar to that of Theorem 5.8. Assume that $[\![\cdot]\!]$ is reasonable; consider the process $P \triangleq (x,y).\mathbf{if}\ x = a\ \mathbf{then\ if}\ y = d\ \mathbf{then}\ \Omega$; pick up $c \neq a$ and $d \neq b$; consider the permutation of names σ swapping a with c and b with d; finally, show that $Q \triangleq P \mid \langle a,b\rangle \mid P\sigma \mid \langle c,d\rangle$ is not divergent, while $[\![Q]\!] \Uparrow$. □

6 Conclusion and Related Work

We have studied the expressive power of eight communication primitives, arising from the combination of three features: arity of data, communication medium and presence of pattern-matching. By relying on possibility/impossibility of 'reasonable' encodings, we obtained a clear hierarchy of communication primitives. Notably, LINDA's communication paradigm [17] is at the top of this hierarchy, while the π-calculus is in the middle. A posteriori, this can justify the fact that the former one is usually exploited in actual programming languages [3, 16], where flexibility and expressive power are the driving issues, while the latter one is mostly used for theoretical reasoning.

Related work One of the pioneering works in the study of communication primitives for distributed systems is [19]. There, the expressive power of several "classical" primitives (like test-and-set, compare-and-swap, ...) is studied by associating to every primitive the highest number of parallel processes that can reach a distributed consensus with that primitive, under conditions similar to the 'reasonableness' of our Definition 3.2. It then follows that a primitive with number n is less expressive than every primitive with number m ($> n$): the latter one can solve a problem (i.e. the consensus among m processes) that the former one cannot reasonably solve. This idea is also exploited in [25] to assess the expressive power of the non-deterministic choice in the π-calculus.

In [12], the notion of relative expressive power is used to measure the expressiveness of programming languages. In particular, a simple class of three concurrent constraint languages is studied and organised in a strict hierarchy. The languages have guarded constructs and only differ in the features offered by the guards: a guard is always passed in the less expressive language; a guard is passed only if a given constraint is satisfied by the current knowledge; and, finally, a guard is passed only if a new constraint, that must be atomically added to the knowledge, is consistent with the current knowledge. Very roughly, the last kind of guards can be related to the pattern-matching construct of our calculi, for the possibility of atomically testing and modifying the environment; in both cases, this feature sensibly increases the expressive power of the language.

By the way, the form of pattern-matching considered here is very minimal: only the equality of names can be tested while retrieving a datum. However, many other forms of pattern-matching can be exploited [14], to yield more and more flexible formalisms; some proposals have been investigated from the expressiveness point of view in [32].

Another form of atomic polyadic name matching is presented in [9], but with a different approach w.r.t. ours. Indeed, while in our π_{111} the tuple of names to be matched is in the transmitted/received value (by using a standard π-calculus terminology, the tuple is in the 'object' part of an output/input), in [9] there are composite channel names that must be matched to enable a communication (thus, the tuple is in the 'subject' part of the output/input). This feature enables a nice modelling of distributed and cryptographic process calculi; nevertheless, our LINDA-like pattern-matching is stronger, since the possibility of using both formal and actual fields in a template yield a more flexible form of input actions.

In this paper we have only considered two key features of LINDA, namely shared dataspaces and pattern-matching. Other two relevant features are the **read** operation and the *conditional predicates*. The former one allows us to access the content of a datum without removing it from the dataspace; the latter ones enhance input/read actions with the possibility of activating different execution threads, according to whether a matching tuple is present in the dataspace or not (thus, they are not blocking). In [6] all the possible combination of these features are studied. In particular, the authors develop a way of giving a structured operational semantics to the resulting languages and investigate possible notions of behavioural equivalences. However, to the best of our knowledge, no encodability result has been developed so far for such languages; this opens another possible direction for future work.

Finally, in [7] three different semantics for the output operation are studied in the setting of a simple LINDA-based process calculus: *instantaneous output* (an output prefix immediately unleashes the corresponding tuple in the dataspace), *ordered output* (a reduction is needed to turn an output prefix into the corresponding tuple in the dataspace) and *unordered output* (two reductions are needed to turn an output into an available tuple, i.e. one to send the tuple to the dataspace and another one to make the tuple available in the dataspace). In [7, 8] it is proved that the semantics can be strictly ordered according to their expressive power, with the instantaneous semantics being the most expressive one and the unordered semantics being the less expressive one (actually, the latter semantics yields a no Turing complete language). According to this terminology, the semantics we used in this paper for the output operation is instantaneous; it would be interesting to discover whether our results still hold also under different semantics for the output actions or not.

Future work This paper is one of the first attempts to classify languages according to their communication primitive. As we have already said, a lot of work still remains to be done. For example, it would be interesting to study more concrete languages, maybe by encoding them in one of the calculi presented in this paper. Moreover, other common features (such as synchrony) could be added to the picture. Finally, it would also be interesting to prove stronger properties for the encodings of Section 4, whenever possible; indeed, since we were mostly interested in the impossibility results, we intentionally exploited quite a weak form of 'reasonabless'.

Acknowledgements I would like to thank Rosario Pugliese for his suggestions that improved a first draft of this paper and Catuscia Palamidessi for her encouragements and discussions. Finally, I also thank the anonymous *FoSSaCS'06* referees for their positive attitude and fruitful comments.

References

[1] L. Acciai and M. Boreale. Xpi: A typed process calculus for xml messaging. In *Proc. of FMOODS'05*, volume 3535 of *LNCS*, pages 47–66. Springer, 2005.

[2] R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195(2):291–324, 1998.

[3] K. Arnold, E. Freeman, and S. Hupfer. *JavaSpaces Principles, Patterns and Practice*. Addison-Wesley, 1999.

[4] G. Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA Sophia-Antipolis, May 1992.

[5] A. Brown, C. Laneve, and G. Meredith. πduce: a process calculus with native XML datatypes. In *Proc. of 2nd Int. Workshop on Services and Formal Methods*, volume 3670 of *LNCS*. Springer, 2005.

[6] N. Busi, R. Gorrieri, and G. Zavattaro. A process algebraic view of LINDA coordination primitives. *Theoretical Computer Science*, 192(2):167–199, 1998.

[7] N. Busi, R. Gorrieri, and G. Zavattaro. Comparing three semantics for LINDA-like languages. *Theoretical Computer Science*, 240(1):49–90, 2000.

[8] N. Busi, R. Gorrieri, and G. Zavattaro. On the expressiveness of LINDA coordination primitives. *Information and Computation*, 156(1-2):90–121, 2000.

[9] M. Carbone and S. Maffeis. On the expressive power of polyadic synchronisation in pi-calculus. *Nordic Journal of Computing*, 10(2):70–98, 2003.

[10] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[11] G. Castagna, R. De Nicola, and D. Varacca. Semantic subtyping for the p-calculus. In *Proc. of LICS*, pages 92–101. IEEE Computer Society, 2005.

[12] F. de Boer and C. Palamidessi. Embedding as a tool for language comparison. *Information and Computation*, 108(1):128–157, 1994.

[13] R. De Nicola, D. Gorla, and R. Pugliese. On the expressive power of KLAIM-based calculi. Tech. Rep. 09/2004, Dip. di Informatica, Univ. di Roma "La Sapienza". To appear in *Theor. Comp. Science*. An extended abstract appeared in the *Proc. of EXPRESS'04*, ENTCS 128(2):117–130.

[14] R. De Nicola, D. Gorla, and R. Pugliese. Pattern matching over a dynamic network of tuple spaces. In M. Steffen and G. Zavattaro, editors, *Proc. of FMOODS'05*, volume 3535 of *LNCS*, pages 1–14. Springer, 2005.

[15] C. Ene and T. Muntean. Expressiveness of point-to-point versus broadcast communications. In *Proc. of 12th Symp. on Fundamentals of Computation Theory*, volume 1684 of *LNCS*, pages 258–268. Springer, 1999.

[16] D. Ford, T. Lehman, S. McLaughry, and P. Wyckoff. T Spaces. *IBM Systems Journal*, pages 454–474, August 1998.

[17] D. Gelernter. Generative Communication in LINDA. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[18] D. Gorla. On the relative expressive power of asynchronous communication primitives. In *Proc. of FoSSaCS'06*, volume 3921 of *LNCS*, pages 47–62. Springer, 2006.

[19] M. Herlihy. Impossibility and universality results for wait-free synchronization. In *Proc. of PODC*, pages 276–290. ACM Press, 1988.

[20] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In P. America, editor, *Proc. of ECOOP '91*, volume 512 of *LNCS*, pages 133–147. Springer, 1991.

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

[22] R. Milner. The polyadic π-calculus: A tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993.

[23] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings of ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[24] U. Nestmann and B. C. Pierce. Decoding choice encodings. *Information and Computation*, 163:1–59, 2000.

[25] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[26] J. Parrow. An introduction to the pi-calculus. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier Science, 2001.

[27] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Foundations of Computing. MIT Press, May 2000.

[28] P. Quaglia and D. Walker. On encoding pπ in mπ. In V. Arvind and R. Ramanujam, editors, *Proceedings of FSTTCS '98*, volume 1530 of *LNCS*, pages 42–51. Springer, Dec. 1998.

[29] D. Sangiorgi. Bisimulation in higher-order process calculi. *Information and Computation*, 131:141–178, 1996.

[30] E. Y. Shapiro. Separating concurrent languages with categories of language embeddings. In *Proc. of 23rd Symposium on Theory of Computing*, pages 198–208. ACM Press, 1991.

[31] N. Yoshida. Graph types for monadic mobile processes. In V. Chandru and V. Vinay, editors, *Proc. of FSTTCS '96*, volume 1180 of *LNCS*, pages 371–386. Springer, 1996.

[32] G. Zavattaro. Towards a hierarchy of negative test operators for generative communication. In *Proc. of EXPRESS*, volume 16 of *ENTCS*, 1998.