Name-passing calculi: from fusions to preorders and types

Daniel Hirschkoff, Jean-Marie Madiot, Davide Sangiorgi

Name-passing calculi: from fusions to preorders and types

Daniel Hirschkoff, Jean-Marie Madiot
ENS Lyon, U. de Lyon, CNRS, INRIA, UCBL
{daniel.hirschkoff, jeanmarie.madiot}@ens-lyon.fr

Davide Sangiorgi
University of Bologna and INRIA
davide.sangiorgi@cs.unibo.it

***Abstract*—The fusion calculi are a simplification of the pi-calculus in which input and output are symmetric and restriction is the only binder. We highlight a major difference between these calculi and the pi-calculus from the point of view of types, proving some impossibility results for subtyping in fusion calculi. We propose a modification of fusion calculi in which the name equivalences produced by fusions are replaced by name preorders, and with a distinction between positive and negative occurrences of names. The resulting calculus allows us to import subtype systems, and related results, from the pi-calculus. We examine the consequences of the modification on behavioural equivalence (e.g., context-free characterisations of barbed congruence) and expressiveness (e.g., full abstraction of the embedding of the asynchronous pi-calculus).**

***Index Terms*—process calculus; fusions; types; subtyping;**

I. Introduction

The π-calculus is the paradigmatical name-passing calculus, that is, a calculus where names (a synonym for "channels") may be passed around. Key aspects for the success of the π-calculus are the minimality of its syntax and its expressiveness. Expressiveness comes at a price: often, desirable behavioural properties, or algebraic laws, fail. The reason is that, when employing π-calculus to describe a system, one normally follows a discipline that governs how names can be used. The discipline can be made explicit by means of *types*. Types bring in other benefits, notably the possibility of statically detecting many programming errors. Types are indeed a fundamental aspect of the π-calculus theory, and one of the most important differences between name-passing calculi and process calculi such as CCS in which names may not be passed.

One of the basic elements in type systems for name-passing calculi is the possibility of separating the capabilities for actions associated to a name, e.g., the capability of using a name in input or in output. The control of capabilities has behavioural consequences because it allows one to express constraints on the use of names. For a simple example, consider a process P that implements two distinct services $\mathtt{A}$ and $\mathtt{B}$, accessible using channels a and b that must be communicated to clients of the services. We assume here only two clients, that receive the channels via c_1 and c_2:

$$P \stackrel{\text{def}}{=} (\boldsymbol{\nu} a, b)\left(\overline{c_1}\langle a, b\rangle.\,\overline{c_2}\langle a, b\rangle.\,(\mathtt{A} \mid \mathtt{B})\right) \qquad (1)$$

We expect that outputs at a or b from the clients are eventually received and processed by the appropriate service. But this is not necessarily the case: a malign client can disrupt the expected protocol by simply offering an input at a or b and then throwing away the values received, or forwarding the values to the wrong service. These misbehaviours are ruled out by a capability type system imposing that the clients only obtain the output capability on the names a and b when receiving them from c_1 and c_2. The typing rules are straightforward, and mimic those for the typing of references in imperative languages with subtyping.

Capabilities [1] are at the basis of more complex type systems, with a finer control on names. For instance, type systems imposing constraints on successive usages of the names like usage-based type systems and deadlock-detection systems, session types, and so on [2], [3], [4].

Capabilities are closely related to subtyping. In the example (1), P creates names a and b, and possesses both the input and the output capabilities on them; it however transmits to the clients only a subset of the capabilities (namely the output capability alone). The subset relation on capabilities gives rise to a subtype relation on types. All forms of subtyping for π-calculus or related calculi in the literature require a discipline on capabilities. Subtyping can also be used to recover well-known forms of subtyping in other computational paradigms, e.g., functional languages or object-oriented languages, when an encoding of terms into processes is enhanced with an encoding of types [5].

An interesting family of variants of the π-calculus are — what we call here — the *fusion calculi*: Fusion [6], Update [7], Explicit Fusions [8], Chi [9], Solos [10]. Their beauty is the simplification achieved, with binding removed from the input construct. Thus input prefixing becomes symmetric to output prefixing, and restriction remains as the only binder. The effect of a synchronisation between an output $\overline{a}b.\,P$ and an input $ac.\,Q$ is to fuse the two object names b and c, which are now interchangeable. Thus communications produce, step-by-step, an equivalence relation on names. Different fusion-like calculi differ in the way the name equivalence is handled. The operational theories of these calculi have been widely studied, e.g. [6], [11], [12], [13], [14].

As for the π-calculus (sometimes abbreviated as π in the sequel), however, the expressiveness of fusion calculi makes desirable behavioural properties fail. The same examples for the π-calculus can be used. For instance, the problems of misbehaving clients of the services of (1) remain. Actually, in fusion calculi additional problems arise; for example a client

receiving the two channels a and b along c_i could fuse them using an input $c_i\langle n, n\rangle.\, R$. Now a and b are indistinguishable, and an emission on one of them can reach any of the two services (and if a definition of a service is recursive, a recursive call could be redirected towards the other service).

In the paper we study the addition of types to fusion calculi; more generally, to single-binder calculi, where input is not binding (in fusion calculi, in addition, reductions fuse names). We begin by highlighting a striking difference between π-calculus and fusion calculi, proving some impossibility results for subtyping (and hence for general capability-based type systems, implicitly or explicitly involving subtyping). In the statement of the results, we assume a few basic properties of type systems for name-passing calculi, such as strengthening, weakening and type soundness, and the validity of the ordinary typing rules for the base operators of parallel composition and restriction. These results do not rule out completely the possibility of having subtyping or capabilities in fusion calculi, because of the few basic assumptions we make. They do show, however, that such type systems would have to be more complex than those for ordinary name-passing calculi such as the π-calculus, or require modifications or constraints in the syntax of the calculi.

Intuitively, the impossibility results arise because at the heart of the operational semantics for fusion calculi is an equivalence relation on names, generated through name fusions. In contrast, subtyping and capability systems are built on a preorder relation (be it subtyping, or set inclusion among subsets of capabilities). The equivalence on names forces one to have an equivalence also on types, instead of a preorder.

We propose a solution whose crux is the replacement of the equivalence on names by a preorder, and a distinction on occurrences of names, between 'positive' and 'negative'. In the resulting single-binder calculus, $\pi\mathtt{P}$ ('π with Preorder'), reductions generate a preorder. The basic reduction rule is

$$\overline{c}a.\, P \mid cb.\, Q \longrightarrow P \mid Q \mid a\!\!/\!b \ .$$

The particle $a\!\!/\!b$, called an *arc*, sets a to be above b in the name preorder. Such a process may redirect a prefix at b (which represents a 'positive' occurrence of b) to become a prefix at a. We show that the I/O (input/output) capability systems of the π-calculus can be reused in $\pi\mathtt{P}$, following a generalisation of the typing rules of the π-calculus that takes into account the negative and positive occurrences of names. A better understanding of type systems with subtyping in name-passing calculi is a by-product of this study. For instance, the study suggests that it is essential for subtyping that substitutions produced by communications (in $\pi\mathtt{P}$, the substitutions produced by arcs) only affect the positive occurrences of names.

The modification also brings in behavioural differences. For instance, both in the π-calculus and in $\pi\mathtt{P}$, a process that creates a new name a has the guarantee that a will remain different from all other known names, even if a is communicated to other processes (only the creator of a can break this, by using a in negative position). This is not true in fusion calculi, where the emission of a may produce fusions between a and other names. To demonstrate the proximity with the π-calculus we show that the embedding of the asynchronous π-calculus into $\pi\mathtt{P}$ is fully abstract (full abstraction of the encoding of the π-calculus into fusion calculi fails). We also exhibit an encoding of Explicit Fusions into $\pi\mathtt{P}$, where fusions become bi-directional arcs.

We present two possible semantics for $\pi\mathtt{P}$ that differ on the moment arcs enable substitutions. In the *eager* semantics, arcs may freely act on prefixes; in the *by-need* semantics, arcs act on prefixes only when interactions occur. We provide a characterisation of the reference contextual behavioural equivalence (barbed congruence) as a context-free labelled bisimilarity for the by-need semantics. We also compare and contrast the semantics, both between them and with semantics based on name fusion.

A property of certain fusion calculi (Fusion, Explicit Fusion) is a semantic duality induced by the symmetry between input and output prefixes. In $\pi\mathtt{P}$, the syntax still allows us to swap inputs and outputs, but in general the original and final processes have incomparable behaviours.

We conclude by examining the following syntactic constraint in single-binder calculi: each name, say b, may occur at most once in negative position (this corresponds to input object, as in $ab.\, P$, or to the source of an arc, as in $a\!\!/\!b$). Under this constraint, the two semantics for $\pi\mathtt{P}$, eager and by-need, coincide. In fusion calculi, the constraint allows us to import the π-calculus type systems. The constraint is however rather strong, and, in fusion calculi, breaks the semantic duality between inputs and outputs.

In summary, $\pi\mathtt{P}$, while being syntactically similar to fusion calculi, remains fairly close to the π-calculus (type systems, management of names).

Further related work: Central to $\pi\mathtt{P}$ is the preorder on names, that breaks the symmetry of name equivalence in fusion-like calculi. Another important ingredient for the theory of $\pi\mathtt{P}$ is the distinction between negative and positive occurrences of a name. In Update [7] and (asymmetric versions of) Chi [9], reductions produce ordinary substitutions on names. In practice, however, substitutions are not much different from fusions: a substitution $\{a/b\}$ fuses a with b *and* makes a the representative of the equivalence class. Still, substitutions are directed, and in this sense Update and Chi look closer to $\pi\mathtt{P}$ than the other fusion calculi. For instance Update and Chi, like $\pi\mathtt{P}$, lack the duality property on computations. Update was refined to the Fusion calculus [6] because of difficulties in the extension with polyadicity. Another major difference for Update and Chi with respect to $\pi\mathtt{P}$ is that in the former calculi substitutions replace all occurrences of names, whereas $\pi\mathtt{P}$ takes into account the distinction between positive and negative occurrences.

The question of controlling the fusion of private names has been addressed in [15], in the U-calculus. This calculus makes no distinction between input and output, and relies on two forms of binding to achieve a better control of scope extrusion, thus leading to a sensible behavioural theory that encompasses fusions and π. Thus the calculus is not single-binder. It is

unclear how capability types could be defined in it, as it does not have primitive constructs for input and output.

Paper outline: Section II gives some background. In Section III, we present some impossibility results on type systems for fusion-like calculi. Section IV introduces $\pi\mathtt{P}$ and its type system. The behavioural theory of $\pi\mathtt{P}$ is explored in Section V, and we give some expressiveness results in Section VI. Section VII studies a syntactical restriction that can be applied to $\pi\mathtt{P}$ and fusions, and we discuss future work in Section VIII.

II. Background on name-passing calculi

In this section we group terminology and notation that are common to all the calculi discussed in the paper. For simplicity of presentation, all calculi in the paper are finite. The addition of operators like replication for writing infinite behaviours goes as expected. The results in the paper would not be affected.

We informally call *name-passing* the calculi in the π-calculus tradition, which have the usual constructs of parallel composition and restriction, and in which computation is interaction between input and output constructs. *Names* identify the pairs of matching inputs/outputs, and the values transmitted may themselves be names. Restriction is a binder for the names; in some cases the input may be a binder too. Examples of these calculi are the π-calculus, the asynchronous π-calculus, the Join calculus, the Distributed π-calculus, the Fusion calculus, and so on. Binders support the usual alpha-conversion mechanism, and give rise to the usual definitions of free and bound names.

Convention 1. To simplify the presentation, throughout the paper, in all statements (including rules), we assume that the bound names of the entities in the statements are *different from each other and different from the free names* (*Barendregt convention on names*). Similarly, we say that a name is *fresh* or *fresh for a process*, if the name does not appear in the entities of the statements or in the process. □

We use $a, b, \ldots$ to range over names. In a free input $ab.\,P$, bound input $a(b).\,P$, output $\overline{a}b.\,P$, we call a the *subject* of the prefix, and b the *object*. We sometimes abbreviate prefixes as $a.\,P$ and $\overline{a}.\,P$ when the object carried is not important. We omit trailing $\mathbf{0}$, for instance writing $\overline{a}b$ in place of $\overline{a}b.\,\mathbf{0}$. We write $P\{a/b\}$ for the result of applying the substitution of b with a in P.

When restriction is the only binder (hence the input construct is not binding), we say that the calculus *has a single binder*. If in addition interaction involves fusion between names, so that we have ($\Longrightarrow$ stands for an arbitrary number of reduction steps, and in the right-hand side P and Q can be omitted if they are $\mathbf{0}$)

$$(\boldsymbol{\nu}c)\,(\overline{a}b.\,P \mid ac.\,Q \mid R) \Longrightarrow (P \mid Q \mid R)\{b/c\}\ , \qquad (2)$$

we say that the calculus *has name-fusions*, or, more briefly, has *fusions*. (We are not requiring that (2) is among the rules of the operational semantics of the calculus, just that (2) holds. The shape of (2) has been chosen so to capture the existing calculi; the presence of R allows us to capture also the Solos calculus.) All single-binder calculi in the literature (Update [7], Chi [9], Fusion [6], Explicit Fusion calculus [11], Solos [10]) have fusions. In Section IV we will introduce a single-binder calculus without fusions.

In all calculi in the paper, (reduction-closed) barbed congruence will be our reference behavioural equivalence. Its definition only requires a reduction relation, $\longrightarrow$, and a notion of barb on names, $\downarrow_a$. Intuitively, a barb at a holds for a process if that process can accept an offer of interaction at a from its environment. We omit the definition, which is standard. We write $\simeq_{\mathcal{L}}$ for (strong) reduction-closed barbed congruence in a calculus $\mathcal{L}$. Informally, $\simeq_{\mathcal{L}}$ is the largest relation that is context-closed, barb-preserving, and reduction-closed. Its weak version, written $\cong_{\mathcal{L}}$, replaces the relation $\longrightarrow_{\mathcal{L}}$ with its reflexive and transitive closure $\Longrightarrow_{\mathcal{L}}$, and the barbs $\downarrow_a^{\mathcal{L}}$ with the weak barbs $\Downarrow_a^{\mathcal{L}}$, where $\Downarrow_a^{\mathcal{L}}$ is the composition of the relations $\Longrightarrow_{\mathcal{L}}$ and $\downarrow_a^{\mathcal{L}}$ (i.e., the barb is visible after some internal actions). See [23] for more details.

III. Typing and subtyping with fusions

We consider typed versions of languages with fusions. We show that in such languages it is impossible to have a non-trivial subtyping, assuming a few simple and standard typing properties of name-passing calculi.

We use T, U to range over types, and Γ to range over type environments, i.e., partial functions from names to types. We write $\mathrm{dom}(\Gamma)$ for the set of names on which Γ is defined. In name-passing calculi, a type system assigns a type to each name. Typing judgements are of the form $\Gamma \vdash P$ (process P respects the type assignments in Γ), and $\Gamma \vdash a : T$ (name a can be assigned type T in Γ).[1] The following are the standard typing rules for parallel composition and restriction:

$$\frac{\Gamma \vdash P_1 \qquad \Gamma \vdash P_2}{\Gamma \vdash P_1 \mid P_2} \qquad \frac{\Gamma, x : T \vdash P}{\Gamma \vdash (\boldsymbol{\nu}x : T)\,P} \qquad (3)$$

The first rule says that any two processes typed in the same type environment can be composed in parallel. The second rule handles name restriction.[2]

In name-passing calculi, the basic type construct is the channel (or connection) type $\sharp\, T$. This is the type of a name that may carry, in an input or an output, values of type T. Consequently, we also assume that the following rule for prefixes $ab.\,P$ and $\overline{a}b.\,P$ is *admissible*.

$$\frac{\Gamma(a) = \sharp\, T \qquad \Gamma(b) = T \qquad \Gamma \vdash P}{\Gamma \vdash \alpha.\,P}\ \alpha \in \{ab, \overline{a}b\} \qquad (4)$$

(Prefixes may not have a continuation, in which case P would be missing from the rule.) In the rule, the type of the subject

[1] We consider in this paper basic type systems and basic properties for them; more sophisticated type systems exist where processes have a type too, e.g., behavioural type systems.

[2] In resource-sensitive type systems, i.e., those for linearity [16] and receptiveness [5], where one counts certain occurrences of the names, the rule for parallel composition has to be modified. As mentioned earlier, in this paper we stick to basic type systems, ignoring resource consumption.

and of the object of the prefix are compatible. Again, these need not be the typing rules for prefixes; we are just assuming that the rules are valid in the type system. The standard rule for prefix would have, as hypotheses,

$$\Gamma \vdash a : \sharp\, T \qquad \Gamma \vdash b : T \ .$$

These imply, but are not equivalent to, the hypotheses in (4), for instance in presence of subtyping.

Fundamental properties of type systems are:

- Subject Reduction (or Type Soundness): if $\Gamma \vdash P$ and $P \to P'$, then $\Gamma \vdash P'$;
- Weakening: if $\Gamma \vdash P$ and a is fresh, then $\Gamma, a : T \vdash P$;
- Strengthening: whenever $\Gamma, a : T \vdash P$ and a is fresh for P, then $\Gamma \vdash P$;
- Closure under injective substitutions: if $\Gamma, a : T \vdash P$ and b is fresh, then $\Gamma, b : T \vdash P\{b/a\}$.

Definition 2. *A typed calculus with single binder is* plain *if it satisfies Subject Reduction, Weakening, Strengthening, Closure under injective substitutions, and the typing rules (3) and (4) are admissible.*

If the type system admits subtyping, then another fundamental property is narrowing, which authorises, in a typing environment, the specialisation of types:

- (Narrowing): if $\Gamma, a : T \vdash P$ and $U \leq T$ then also $\Gamma, a : U \vdash P$.

When narrowing holds, we say that the calculus *supports narrowing*.

A typed calculus *has trivial subtyping* if, whenever $T \leq U$, we have $\Gamma, a : T \vdash P$ iff $\Gamma, a : U \vdash P$. When this is not the case (i.e., there are T, U with $T \leq U$, and T, U are not interchangeable in all typing judgements) we say that the calculus has *meaningful* subtyping.

Under the assumptions of Definition 2, a calculus with fusions may only have trivial subtyping.

Theorem 3. *A typed calculus with fusions that is plain and supports narrowing has trivial subtyping.*

In the proof, given in [23], we assume a meaningful subtyping and use it to derive a contradiction from type soundness and the other hypotheses.

One may wonder whether, in Theorem 3, more limited forms of narrowing, or a narrowing in the opposite direction, would permit some meaningful subtyping. Narrowing is interesting when it allows us to modify the type of the values exchanged along a name, that is, the type of the object of a prefix. (In process calculi, communication is the analogous of application for functional languages, and changing the type of an object is similar to changing the type of a function or of its argument.) In other words, disallowing narrowing on objects would make subtyping useless. We show that *any* form of narrowing, on one prefix object, would force subtyping to be trivial.

Theorem 4. *Suppose a typed calculus with fusions is plain and there is at least one prefix α with object b, different from the subject, and there are two types S and T such that $S \leq T$ and one of the following forms of narrowing holds for all Γ:*

1) *whenever $\Gamma, b : T \vdash \alpha.\,\mathbf{0}$, we also have $\Gamma, b : S \vdash \alpha.\,\mathbf{0}$;*
2) *whenever $\Gamma, b : S \vdash \alpha.\,\mathbf{0}$, we also have $\Gamma, b : T \vdash \alpha.\,\mathbf{0}$.*

Then S and T are interchangeable in all typing judgements.

As a consequence, authorising one of the above forms of narrowing for all S and T such that $S \leq T$ implies that the calculus has trivial subtyping. The proof of Theorem 4 is similar to that of Theorem 3.

Remark 5. Theorems 3 and 4 both apply to all fusion calculi: Fusion, Explicit Fusions, Update, Chi, Solos (where the continuation P is $\mathbf{0}$). □

Another consequence of Theorems 3 and 4 is that it is impossible, in plain calculi with fusions, to have an I/O type system; more generally, it is impossible to have any capability-based type system that supports meaningful subtyping.

Actually, to apply the theorems, it is not even necessary for the capability type system to have an explicit notion of subtyping. For Theorem 3, it is sufficient to have sets of capabilities with a non-trivial ordering under inclusion, meaning that we can find two capability types T and U such that whenever $\Gamma, a : U \vdash P$ holds then also $\Gamma, a : T \vdash P$ holds, but not the converse (e.g., T provides more capabilities than U). We could then impose a subtype relation $\leq$ on types, as the least preorder satisfying $T \leq U$. Theorem 3 then tells us that type soundness and the other properties of Definition 2 would require also $U \leq T$ to hold, i.e., T and U are interchangeable in all typing judgements. In other words, the difference between the capabilities in T and U has no consequence on typing. Similarly, to apply Theorem 4 it is sufficient to find two capability types T and U and a single prefix in whose typing U can replace T.

IV. A CALCULUS WITH NAME PREORDERS

A. Preorders, positive and negative occurrences

We now refine the fusion calculi by replacing the equivalence relation on names generated through communication by a preorder, yielding $\pi\mathtt{P}$ ('π with Preorder'). As the preorder on types given by subtyping allows promotions between related types, so the preorder on names of $\pi\mathtt{P}$ allows promotions between related names. Precisely, if a is below a name b in the preorder, then a prefix at a may be promoted to a prefix at b and then interact with another prefix at b. Thus an input $av.\,P$ may interact with an output $\overline{b}w.\,Q$; and, if also c is below b, then $av.\,P$ may as well interact with an output $\overline{c}z.\,R$.

The ordering on names is introduced by means of the *arc* construct, $a\!\!/\!\!b$, that declares the *source* b to be below the *target* a. The remaining operators are as for fusion calculi (i.e., those of the π-calculus with bound input replaced by free input).

$$P ::= \mathbf{0} \ \Big|\ P \mid P \ \Big|\ \overline{a}b.\,P \ \Big|\ ab.\,P \ \Big|\ \boldsymbol{\nu} aP \ \Big|\ a\!\!/\!\!b \ .$$

The semantics of the calculus is given in the reduction style. Structural congruence, $\equiv$, is defined as the usual congruence produced by the monoidal rules for parallel composition and

the rules for commuting and extruding restriction (see [23] for a complete definition). We explain the effect of reduction by means of contexts, rather than separate rules for each operator. Contexts allow us a more succinct presentation, and a simpler comparison with an alternative semantics (Section V). An *active context* is one in which the hole may reduce. Thus the only difference with respect to ordinary contexts is that the hole may not occur underneath a prefix. We use C to range over (ordinary) contexts, and E for active contexts. The rules for reduction are as follows (the subscript in $\longrightarrow_{\mathrm{ea}}$, for "eager", will distinguish this from the alternative semantics in Section V-A):

$$\text{R-SCon}: \quad \frac{P \equiv E[Q] \qquad Q \longrightarrow_{\mathrm{ea}} Q' \qquad E[Q'] \equiv P'}{P \longrightarrow_{\mathrm{ea}} P'}$$

$$\text{R-Inter}: \quad \overline{a}b.\,P \mid ac.\,Q \longrightarrow_{\mathrm{ea}} P \mid Q \mid b\!\not\!c$$

$$\text{R-SubOut}: \quad a\!\not\!b \mid \overline{b}c.\,Q \longrightarrow_{\mathrm{ea}} a\!\not\!b \mid \overline{a}c.\,Q$$

$$\text{R-SubInp}: \quad a\!\not\!b \mid bc.\,Q \longrightarrow_{\mathrm{ea}} a\!\not\!b \mid ac.\,Q$$

Rule R-Inter shows that communication generates an arc. Rules R-SubOut and R-SubInp show that arcs only act on the subject of prefixes; moreover, they only act on *unguarded* prefixes (i.e., prefixes that are not underneath another prefix). The rules also show that arcs are persistent processes. Acting only on prefix subjects, arcs can be thought of as particles that "redirect prefixes": an arc $a\!\not\!b$ redirects a prefix at b towards a higher name a. (Arcs remind us of special π-calculus processes, called forwarders or wires [17], which under certain hypotheses allow one to model substitutions; as for arcs, so the effect of forwarders is to replace the subject of prefixes.)

We write $\Longrightarrow_{\mathrm{ea}}$ for the reflexive and transitive closure of $\longrightarrow_{\mathrm{ea}}$. Here are some examples of reduction.

$$\begin{array}{rrl}
 & & \overline{a}c.\overline{c}a.e.P \mid ad.de.\overline{a}.Q \\
\text{R-Inter} & \longrightarrow_{\mathrm{ea}} & \overline{c}a.e.P \mid \quad de.\overline{a}.Q \mid c\!\not\!d \\
\text{R-SubInp} & \longrightarrow_{\mathrm{ea}} & \overline{c}a.e.P \mid \quad ce.\overline{a}.Q \mid c\!\not\!d \\
\text{R-Inter} & \longrightarrow_{\mathrm{ea}} & e.P \mid \quad \overline{a}.Q \mid c\!\not\!d \mid a\!\not\!e \\
\text{R-SubInp} & \longrightarrow_{\mathrm{ea}} & a.P \mid \quad \overline{a}.Q \mid c\!\not\!d \mid a\!\not\!e \\
\text{R-Inter} & \longrightarrow_{\mathrm{ea}} & P \mid \quad Q \mid c\!\not\!d \mid a\!\not\!e
\end{array}$$

Reductions can produce multiple arcs that act on the same name. This may be used to represent certain forms of choice, as in the following processes:

$$\begin{array}{l}
(\boldsymbol{\nu} h, k)\,(bu.\,cu.\,\overline{u} \mid \overline{b}h.\,h.\,P \mid \overline{c}k.\,k.\,Q) \\
\Longrightarrow_{\mathrm{ea}}\ (\boldsymbol{\nu} h, k)\,(\overline{u} \mid h\!\not\!u \mid k\!\not\!u \mid h.\,P \mid k.\,Q)\ .
\end{array}$$

Both arcs may act on $\overline{u}$, and are therefore in competition with each other. The outcome of the competition determines which process between P and Q is activated. For instance, reduction may continue as follows:

$$\begin{array}{rrl}
\text{R-SubOut} & \longrightarrow_{\mathrm{ea}} & (\boldsymbol{\nu} h, k)\,(\overline{k} \mid h\!\not\!u \mid k\!\not\!u \mid h.\,P \mid k.\,Q) \\
\text{R-Inter} & \longrightarrow_{\mathrm{ea}} & (\boldsymbol{\nu} h, k)\,(h\!\not\!u \mid k\!\not\!u \mid h.\,P \mid Q)\ .
\end{array}$$

Definition 6 (Positive and negative occurrences)**.** *In an input $ab.\,P$ and an arc $a\!\not\!b$, the name b has a* negative occurrence*. All other occurrences of names in input, output and arcs are* positive occurrences*.*

An occurrence in a restriction ($\boldsymbol{\nu}$a) is neither negative nor positive, intuitively because restriction acts only as a binder, and does not stand for an usage of the name (in particular, it does not take part in a substitution).

Negative occurrences are particularly important, as by properly tuning them, different usages of names may be obtained. For instance, a name with zero negative occurrence is a constant (i.e., it is a channel, and may not be substituted); and a name that has a single negative occurrence is like a π-calculus name bound by an input (see Section VI-B).

The number of negative occurrences of a name is invariant under reduction.

Lemma 7. *If $P \longrightarrow_{\mathrm{ea}} P'$ then for each b, the number of negative occurrences of b in P and P' is the same.*

B. Types

We now show that the I/O capability type system and its subtyping can be transplanted from π to $\pi\mathtt{P}$. In all typed calculi in the paper, binding occurrences of names are annotated with their type — we are not concerned with type inference.

In the typing rules for I/O-types in the (monadic) π-calculus [1], two additional types are introduced: $\mathtt{o}\,T$, the type of a name that can be used only in output and that carries values of type T; and $\mathtt{i}\,T$, the type of a name that can be used only in input and that carries values of type T. The subtyping rules stipulate that $\mathtt{i}$ is covariant, $\mathtt{o}$ is contravariant, and $\sharp$ is invariant. Subtyping is brought up into the typing rules through the subsumption rule. The most important typing rules are those for input and output prefixes; for input we have:

$$\text{T-InpBound}: \quad \frac{\Gamma \vdash a : \mathtt{i}\,T \qquad \Gamma, b : T \vdash P}{\Gamma \vdash a(b : T).\,P}$$

The π-calculus supports narrowing, and this is essential in the proof of subject reduction.

The type system for $\pi\mathtt{P}$ is presented in Table I. With respect to the π-calculus, only the rule for input needs an adjustment, as $\pi\mathtt{P}$ uses free, rather than bound, input. The idea in rule T-InpFree of $\pi\mathtt{P}$ is however the same as in rule T-InpBound of π: we look up the type of the object of the prefix, say T, and we require $\mathtt{i}\,T$ as the type for the subject of the prefix. To understand the typing of an arc $a\!\not\!b$, recall that such an arc allows one to replace b with a. Rule T-Arc essentially checks that a has at least as many capabilities as b, in line with the intuition for subtyping in capability type systems.

Common to all premises of T-InpBound, T-InpFree and T-Arc is the look-up of the type of names that occur negatively (the source of an arc and the object of an input prefix): the type that appears for b in the hypothesis is precisely the type found in the conclusion (within the process or in Γ). In contrast, the types for positive occurrences may be different (e.g., because of subsumption $\Gamma \vdash a : \mathtt{i}\,T$ may hold even if $\Gamma(a) \neq \mathtt{i}\,T$). We cannot type inputs like outputs: consider

$$\text{T-InpFree2-Wrong}: \quad \frac{\Gamma \vdash a : \mathtt{i}\,T \qquad \Gamma \vdash b : T}{\Gamma \vdash ab}$$

Rule T-InpFree2-Wrong would accept, for instance, an input ab in an environment Γ where $a : \mathtt{i}\,\mathtt{i}\,\mathbf{1}$ and $b : \sharp\,\mathbf{1}$. By

Types ($\mathbf{1}$ is the unit type): $\quad T ::= \mathtt{i}\,T \mid \mathtt{o}\,T \mid \sharp\,T \mid \mathbf{1}$

Subtyping rules:

$$\frac{}{\sharp\,T \leq \mathtt{i}\,T} \qquad \frac{}{\sharp\,T \leq \mathtt{o}\,T} \qquad \frac{S \leq T}{\mathtt{i}\,S \leq \mathtt{i}\,T} \qquad \frac{S \leq T}{\mathtt{o}\,T \leq \mathtt{o}\,S} \qquad \frac{}{T \leq T} \qquad \frac{S \leq T \qquad T \leq U}{S \leq U}$$

Typing rules:

$$\textsc{Tv-Name}\ \frac{}{\Gamma, a : T \vdash a : T} \qquad \textsc{Subsumption}\ \frac{\Gamma \vdash a : S \qquad S \leq T}{\Gamma \vdash a : T} \qquad \textsc{T-Res}\ \frac{\Gamma, a : T \vdash P}{\Gamma \vdash \boldsymbol{\nu} a P} \qquad \textsc{T-Par}\ \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad \textsc{T-Nil}\ \frac{}{\Gamma \vdash \mathbf{0}}$$

$$\textsc{T-Out}\ \frac{\Gamma \vdash a : \mathtt{o}\,T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P}{\Gamma \vdash \overline{a} b.\, P} \qquad \textsc{T-InpFree}\ \frac{\Gamma \vdash a : \mathtt{i}\,\Gamma(b) \qquad \Gamma \vdash P}{\Gamma \vdash a b.\, P} \qquad \textsc{T-Arc}\ \frac{\Gamma \vdash a : \Gamma(b)}{\Gamma \vdash a/\!\!/b}$$

TABLE I
THE TYPE SYSTEM OF $\pi\mathtt{P}$

subtyping and subsumption, we could then derive $\Gamma \vdash b : \mathtt{i}\,\mathbf{1}$. In contrast, rule T-InpFree, following the input rule of the π-calculus, makes sure that the object of the input does not have too many capabilities with respect to what is expected in the type of the subject of the input. This constraint is necessary for subject reduction. As a counterexample, assuming rule T-InpFree2-Wrong, we would have $a : \sharp\,\mathtt{i}\,\mathbf{1}, b : \sharp\,\mathbf{1}, c : \mathtt{i}\,\mathbf{1} \vdash P$, for $P \stackrel{\text{def}}{=} ab \mid \overline{a}c \mid \overline{b}$. However, $P \longrightarrow_{\text{ea}} c/\!\!/b \mid \overline{b} \longrightarrow_{\text{ea}} c/\!\!/b \mid \overline{c}$, and the final derivative is not typable under Γ (as Γ only authorises inputs at c).

In $\pi\mathtt{P}$, the direction of the narrowing is determined by the negative or positive occurrences of a name.

Theorem 8 (Polarised narrowing)**.** *Let T_1 and T_2 be two types such that $T_1 \leq T_2$.*

1) *If a occurs only positively in P, then $\Gamma, a : T_2 \vdash P$ implies $\Gamma, a : T_1 \vdash P$.*
2) *If a occurs only negatively in P, then $\Gamma, a : T_1 \vdash P$ implies $\Gamma, a : T_2 \vdash P$.*
3) *If a occurs both positively and negatively in P, then it is in general unsound to replace, in a typing $\Gamma \vdash P$, the type of a in Γ with a subtype or supertype.*

Theorem 8 (specialised to prefixes) does not contradict Theorem 4, because in $\pi\mathtt{P}$, reduction does not satisfy (2) (from Section II). Our system enjoys subject reduction:

Theorem 9. *If $\Gamma \vdash P$ and $P \longrightarrow_{\text{ea}} P'$ then also $\Gamma \vdash P'$.*

Remark 10. Theorem 8 may be seen as a refinement of the standard narrowing result for name-passing calculi. In the π-calculus, for instance, a free name only has positive occurrences. Hence the usual narrowing corresponds to Theorem 8(1). And in an input $a(b : T).\, P$, the binder for b represents a negative occurrence, so that if b is free in P then b has both positive and negative occurrences, which means that the type T may not be modified, as by Theorem 8(3). In contrast, Theorem 8(2) is vacuous in π, as a name b with only negative occurrences is found in an input $a(b : T).\, P$ where b is not free in P.

In general, in a name-passing calculus, if a name has only positive occurrences, then its type (be it declared in the typing environment, or in the binding occurrence of that name within the process) may be replaced by a subtype, and conversely for names with only negative occurrences, whereas the type of names with both positive and negative occurrences may not be changed. Defining rules that distinguish between negative and positive occurrences in name-passing calculi is beyond the scope of this paper. A rule of thumb however seems that if the occurrence of a name generates a substitution acting on that name (i.e., a replacement of the name), then the occurrence is negative; if it does not, then it is positive. Thus in a fusion $a = b$ of the Explicit Fusion calculus, the occurrences of a and b are both positive and negative, as a fusion may produce a substitution $a/\!\!/b$ or a substitution $b/\!\!/a$ (which, incidentally, gives another explanation of the impossibility of narrowing in presence of an explicit fusion construct). □

Remark 11. For the Subject Reduction theorem for $\pi\mathtt{P}$ it is critical that an arc $a/\!\!/b$ only acts on positive occurrences of b. Provided this is respected, the theorem remains valid under different behaviours for arcs (e.g., simultaneously replacing all positive occurrences of b, not only at top-level). □

V. Behaviours

A. An alternative semantics

The operational semantics given to $\pi\mathtt{P}$ in Section IV allows arcs to act locally, at any time. The effect of an arc is irreversible: the application of an arc $a/\!\!/b$ to a prefix at b commits that prefix to interact along a name that is greater than, or equal to, a in the preorder among names. A commitment may disable certain interactions, even block a prefix for ever. Consider, e.g.,

$$(\boldsymbol{\nu} a, c)\ (bv.\, P \mid \overline{c}w.\, Q \mid a/\!\!/b \mid c/\!\!/b) \tag{5}$$

There is a competition between the two arcs; if the first wins, the process is deadlocked:

$$\longrightarrow_{\mathrm{ea}} (\boldsymbol{\nu} a, c)\,(av.\,P \mid \overline{c}w.\,Q \mid a/b \mid c/b)$$

since a and c are unrelated in the preorder.

We consider here an alternative semantics, in which the action of arcs is not a commitment: arcs come about only when interaction occurs. For this reason we call the new semantics *by-need* (arcs act only when 'needed'), whereas we call *eager* the previous semantics (arcs act regardless of matching prefixes). In this semantics, as in the π-calculus, an interaction involves both a synchronisation and a substitution; however unlike in the π-calculus where the substitution is propagated to the whole term, here substitution only replaces the subject of the interacting prefixes.

The formalisation of the new semantics makes use of the partial order on names induced by arcs. In a process, an arc is *active* if it is unguarded, i.e., it is not underneath a prefix. We write $\mathtt{preor}(P)$ for the preorder on names produced by the active arcs in P (i.e., the least preorder $\leq$ that includes $b \leq a$ for each active arc a/b in P). Similarly, $\mathtt{preor}(C)$ is the preorder produced by the active arcs of the context C. Note that this definition relies on the Barendregt convention on names (Convention 1), as it is purely syntactic, i.e., if P and P' are alpha convertible then $\mathtt{preor}(P)$ and $\mathtt{preor}(P')$ may be different. A definition that does not rely on the convention is given in [23].

We write $P \triangleright a \curlyvee b$ if $\{a, b\}$ has an upper bound in the preorder $\mathtt{preor}(P)$, that is, there is a name that is above both a and b; in this case we also say that a and b are *joinable*. Similarly we write $C \triangleright a \curlyvee b$ for contexts. For instance, we have $\boldsymbol{\nu} u(u/a \mid u/b \mid Q) \triangleright a \curlyvee b$, and $\boldsymbol{\nu} v(\overline{v}t \mid (\boldsymbol{\nu} w)(w/v \mid a/w \mid [\cdot]) \triangleright a \curlyvee v$. We have $P \triangleright a \curlyvee b$ iff $P' \triangleright a \curlyvee b$ if P and P' are alpha convertible and a and b occur free in P.

Example 12. *A process $M_{fg} = (\boldsymbol{\nu} c)(c/f \mid c/g)$ acts like a* mediator*: it joins names f and g (we have $M_{fg} \triangleright f \curlyvee g$). Mediators remind us of equators in the π-calculus, or of fusions in the Explicit Fusion calculus, but lack the transitivity property (e.g., $M_{fg} \mid M_{gh} \triangleright f \curlyvee h$ does not hold).*

Definition 13 (By-need reduction)**.** *The* by-need *reduction relation, $P \longrightarrow_{\mathrm{bn}} P'$, is defined by the following rules, where $\equiv$ is as in the eager semantics:*

$$\text{BN-SCON}: \quad \frac{P \equiv E[Q] \qquad Q \longrightarrow_{\mathrm{bn}} Q' \qquad E[Q'] \equiv P'}{P \longrightarrow_{\mathrm{bn}} P'}$$

$$\text{BN-RED}: \quad \frac{E \triangleright a \curlyvee b}{E[ac.\,P \mid \overline{b}d.\,Q] \longrightarrow_{\mathrm{bn}} E[P \mid d/c \mid Q]}$$

Relation $\Longrightarrow_{\mathrm{bn}}$ is the reflexive transitive closure of $\longrightarrow_{\mathrm{bn}}$.

While the eager semantics has simpler rules, the by-need semantics avoids 'too early commitments' on prefixes. For instance, the only immediate reduction of the process in (5) is

$$\longrightarrow_{\mathrm{bn}} (\boldsymbol{\nu} a, c)\,(P \mid w/v \mid Q \mid a/b \mid c/b)$$

where prefixes $bv.\,P$ and $\overline{c}w.\,Q$ interact because their subjects are joinable in the preorder generated by the two arcs.

Lemma 14 (Eager and by-need)**.** *$P \longrightarrow_{\mathrm{bn}} P'$ (by-need semantics) implies $P \Longrightarrow_{\mathrm{ea}} P'$ (eager semantics).*

Corollary 15. *Theorem 9 holds for the by-need semantics.*

B. Behavioural equivalence

We contrast barbed congruence in $\pi\mathtt{P}$ under the two semantics we have given, eager and by-need. We have already defined reduction relations, we only need to define barbs. This requires some care, as the interaction of a process with its environment may be mediated by arcs. For this, and to have a uniform definition of barbs under the eager and by-need semantics, we follow the definition of success in testing equivalence [18], using a special signal ω that we assume may not appear in processes: thus for any name a, the barb $\downarrow_a$ holds for a process P if there is a prefix α with subject a such that $P \mid \alpha.\,\omega$ reduces in one step to a process in which ω is unguarded (i.e., the offer of the environment of an action at a may be accepted by P). Weak barbs and barbed congruence are then defined in the standard way, as outlined in Section II. We write $\simeq_{\mathrm{ea}}$ and $\cong_{\mathrm{ea}}$ (resp. $\simeq_{\mathrm{bn}}$ and $\cong_{\mathrm{bn}}$) for the strong and weak versions of eager (resp. by-need) barbed congruence.

The eager and by-need semantics of $\pi\mathtt{P}$ yield incomparable equivalences. The two following laws are valid in the by-need case, and fail in the eager case:

$$(\boldsymbol{\nu} a)a/c = \mathbf{0} \qquad\qquad a \mid a = a.\,a \ .$$

To see the failure of the first law in the eager semantics, consider a context $C \stackrel{\text{def}}{=} [\cdot] \mid (\boldsymbol{\nu} b)(b/c) \mid c \mid \overline{c}.\,\overline{w}$; then $C[(\boldsymbol{\nu} a)(a/c)]$ can lose the possibility of emitting at w, by reducing in two steps to $(\boldsymbol{\nu} a)(a/c \mid a) \mid (\boldsymbol{\nu} b)(b/c \mid \overline{b}.\,\overline{w})$, because of a commitment determined by arcs; this cannot happen for $C[\mathbf{0}]$. There are no early commitments in the by-need semantics, for which the two processes are hence equal.

Similarly, in the eager semantics, it is possible to put $a \mid a$ in a context where two arcs rewrite each a prefix differently, while one can only rewrite the topmost prefix in $a.\,a$. This scenario cannot be played in the by-need semantics.

On the other hand, the following law is valid for strong (and weak) eager equivalence, but fails to hold in the by-need case:

$$(\boldsymbol{\nu} abu)(a/u \mid b/u \mid \overline{u} \mid a.\,\overline{w}) = (\boldsymbol{\nu} v)(\overline{v} \mid v.\,\tau.\,\overline{w} \mid v.\,\mathbf{0}) \ .$$

($\tau.\,\overline{w}$ stands for $\boldsymbol{\nu} c(c \mid \overline{c}.\,\overline{w})$). The intuition is that concurrent substitutions are used on the left-hand side to implement internal choice. As a consequence of the law $(\boldsymbol{\nu} a)a/c = \mathbf{0}$, in the by-need case, process b/u can be disregarded on the left, so that the process on the left *must* do the output on w.

We have introduced $\pi\mathtt{P}$ with the eager semantics for reasons of simplicity, but we find the by-need semantics more compelling. Below, unless otherwise stated, we work under by-need, though we also indicate what we know under eager.

C. Context-free characterisations of barbed congruence

When it comes to proving behavioural equalities, the definition of barbed congruence is troublesome, as it involves a heavy quantification on contexts. One therefore looks for context-free coinductive characterisations, as labelled bisimilarities that take into account not only reductions within a process, but also the potential interactions between the process and its environment (e.g., input and output actions). We present such characterisation for the by-need equivalence; currently we do not have one for the eager.

As actions for the by-need labelled bisimilarity, we use, besides τ-actions, only free input and free output:

$$\mu \quad ::= \quad \tau \mid ab \mid \overline{a}b \ .$$

In by-need, labelled transitions are written $P \xrightarrow{\mu}_{\mathrm{bn}} P'$. Internal transitions have already been defined, in the reduction semantics, thus we can take relation $\xrightarrow{\tau}_{\mathrm{bn}}$ to coincide with the reduction relation $\longrightarrow_{\mathrm{bn}}$. Input and output transitions are defined by these rules:

$$\text{BN-INP}: \quad \frac{E \triangleright a \curlyvee b \qquad E \text{ does not bind } b \text{ and } d}{E[ac.\, P] \xrightarrow{bd}_{\mathrm{bn}} E[d/c \mid P]}$$

$$\text{BN-OUT}: \quad \frac{E \triangleright a \curlyvee b \qquad E \text{ does not bind } b \text{ and } d}{E[\overline{a}c.\, P] \xrightarrow{\overline{b}d}_{\mathrm{bn}} E[c/d \mid P]}$$

The purpose of the two rules is to define the input and output transitions, with labels as simple as possible, with which to derive a labelled bisimilarity. The two rules are not supposed to be composed together to derive τ-actions (which are computed from the rules of reduction). We leave the definition of a pure SOS semantics, which avoids the structural manipulations of structural congruence, for future work.

To understand rules BN-INP and BN-OUT, suppose the environment is offering an action at b. Since a and b are joinable, there is a name, say e, that is above both a and b in the preorder; hence the prefix at a in the process and the prefix at b in the environment can be transformed into prefixes at e, and can interact. The need for the preorder explains why we found it convenient to express actions via active contexts. In the action, the use of a free object d allows us to ignore name extrusion and thus simplifies the bisimulation checks. As an example of BN-OUT, we have (similar observations can be made for BN-INP):

$$\begin{aligned}&(\boldsymbol{\nu} u)\left(\, u/b \mid (\boldsymbol{\nu} a, c)(u/a \mid \overline{a}c.\, P)\,\right)\\ &\quad\xrightarrow{\overline{b}d}_{\mathrm{bn}}\ (\boldsymbol{\nu} u)\left(\, u/b \mid (\boldsymbol{\nu} a, c)(u/a \mid c/d \mid P)\,\right)\ .\end{aligned}$$

Here the process can interact with the environment at b (and hence perform a transition where b is the subject), because a and b are joinable. Name c is not extruded; instead the arc c/d redirects interactions on d to c.

The labelled bisimulation requires, besides the invariance for actions, invariance under the addition of arcs; moreover a check is made on the visible effects of arcs. In the clause for actions, no extrusion or binding on names is involved; further, it is sufficient that the objects of the actions are *fresh names*.

Definition 16 (Bisimulation)**.** *A* by-need *bisimulation $\mathcal{R}$ is a set of pairs (P, Q) s.t. $P\mathcal{R}Q$ implies:*

1) *$P \mid a/b \ \mathcal{R}\ Q \mid a/b$, for each name a, b (invariance under arcs);*
2) *if a and b appear free in P, then $P \triangleright a \curlyvee b$ implies $Q \triangleright a \curlyvee b$;*
3) *if $P \xrightarrow{\mu}_{\mathrm{bn}} P'$, then $Q \xrightarrow{\mu}_{\mathrm{bn}} Q'$ and $P'\mathcal{R}Q'$ (where the object part of μ is fresh);*
4) *the converse of clauses (2) and (3).*

Bisimilarity, written $\sim_{\mathrm{bn}}$, is the largest bisimulation.

We now present some examples and laws that are proved using the coinductive proof method of labelled bisimilarity. All equalities and inequalities also hold under the eager semantics, though for some equalities only in the weak case (e.g., Lemma 19).

Any input and output of $\pi\mathtt{P}$ can be transformed into a bound prefix, by introducing a new restricted name:

Lemma 17. *We have $ax.\, P \sim_{\mathrm{bn}} (\boldsymbol{\nu} x')ax'.\, (x'/x \mid P)$ and $\overline{b}y.\, Q \sim_{\mathrm{bn}} (\boldsymbol{\nu} y')\overline{b}y'.\, (y/y' \mid Q)$, for fresh x' and y'.*

If these laws are applied to all inputs and outputs of a process P, then the result is a process P' that is behaviourally the same as P, and in which all names exchanged in an interaction are fresh. Thus P' reminds us of a variant of π that achieves symmetry between input and output constructs, namely πI, the π-calculus with internal mobility [19].

Lemma 18. *We have $(\boldsymbol{\nu} b, c)\overline{a}c.\, \overline{a}b.\, \mathbf{0} \not\sim_{\mathrm{bn}} (\boldsymbol{\nu} c)\overline{a}c.\, \overline{a}c.\, \mathbf{0}$, and $(\boldsymbol{\nu} b, c)ac.\, ab.\, \mathbf{0} \sim_{\mathrm{bn}} (\boldsymbol{\nu} c)ac.\, ac.\, \mathbf{0}$.*

These laws show a difference between input and output in behavioural equalities. The reason for the inequality is that the first process can produce two transitions with objects e, f yielding $P \stackrel{\text{def}}{=} \boldsymbol{\nu} c\, (c/f \mid c/e)$, and then $P \triangleright e \curlyvee f$.

Lemma 19 (Substitution and polarities)**.**

1) *If name a has only positive occurrences in P, then $(\boldsymbol{\nu} a)(P \mid b/a) \sim_{\mathrm{bn}} P\{b/a\}$;*
2) *if name a has only negative occurrences in P, then $(\boldsymbol{\nu} a)(P \mid a/b) \sim_{\mathrm{bn}} P\{b/a\}$;*
3) *$(\boldsymbol{\nu} a)(P \mid b/a \mid a/b) \sim_{\mathrm{bn}} P\{b/a\}$.*

For the comparison between labelled bisimilarity and barbed congruence, the most delicate part is the proof of congruence for bisimilarity. This is due to the shape of visible transitions, where an arc is introduced and the object part is always a fresh name, and to the use of $\equiv$ in the definition of transitions. The proof can be found in [23].

Theorem 20. *Bisimilarity is a congruence.*

Theorem 21 (Characterisation of barbed congruence)**.** *In $\pi\mathtt{P}$, relations $\sim_{\mathrm{bn}}$ and $\simeq_{\mathrm{bn}}$ coincide.*

Hence all the laws stated above for $\sim_{\mathrm{bn}}$ hold for $\simeq_{\mathrm{bn}}$.

VI. EXPRESSIVENESS OF $\pi\mathtt{P}$

We compare $\pi\mathtt{P}$ with a few other calculi, both as examples of the use of the calculus and as a test for its expressiveness.

When useful, we work in a *polyadic* version of $\pi\mathtt{P}$; the addition of polyadicity goes as for other name-passing calculi in the literature. All results in this section use the by-need semantics; we do not know their status under the eager semantics.

A. Explicit Fusions

Bi-directional arcs, e.g., $a/b \mid b/a$, work as name fusions (cf, Lemma 19(3)). We thus can encode calculi based on name fusion into $\pi\mathtt{P}$. As an example, we consider the Explicit Fusion calculus [8]. Its syntax extends the Fusion calculus with a fusion construct $a = b$. The encoding is defined as follows for prefixes and explicit fusions, the other constructs being encoded homomorphically:

$$\begin{array}{rcl} [\![\overline{a}\langle v\rangle.\, P]\!] & = & (\boldsymbol{\nu} w)\overline{a}\langle v, w\rangle.\, wv.\, [\![P]\!] \\ [\![ax.\, Q]\!] & = & (\boldsymbol{\nu} y)a\langle x, y\rangle.\, \overline{y}\langle x\rangle.\, [\![Q]\!] \\ [\![a = b]\!] & = & a/b \mid b/a \end{array}$$

In Explicit Fusions, an interaction introduces a name fusion. In the $\pi\mathtt{P}$ encoding, this is mimicked in two steps so to be able to produce bidirectional arcs. The second step is the reverse of the original interaction, and is realised by means of an extra private name. We have operational correspondence for the encoding (we do not know whether it is is fully abstract).

Theorem 22. *Let P, Q be processes of the Explicit Fusion calculus, and $\longrightarrow_{\mathrm{EF}}$ the reduction relation in the calculus.*

1) *If $P \equiv Q$ then $[\![P]\!] \simeq_{\mathrm{bn}} [\![Q]\!]$;*
2) *if $P \longrightarrow_{\mathrm{EF}} P'$ then $[\![P]\!] \longrightarrow_{\mathrm{bn}} \cong_{\mathrm{bn}} [\![P']\!]$;*
3) *conversely, if $[\![P]\!] \longrightarrow_{\mathrm{bn}} Q$, then $Q \cong_{\mathrm{bn}} [\![P']\!]$ for some P' such that $P \longrightarrow_{\mathrm{EF}} P'$.*

A similar result holds for the Fusion calculus, though for Explicit Fusions the statement is simpler because in the latter calculus a restriction is not necessary for fusions to act.

B. π-calculus

The embedding of the π-calculus into a fusion calculus is defined by translating the bound input construct as follows:

$$[\![a(x).\, P]\!] = (\boldsymbol{\nu} x)\, ax.\, [\![P]\!]$$

(the other constructs being translated homomorphically). The same encoding can be used for $\pi\mathtt{P}$.

The encoding of π-calculus into Fusions is not fully abstract for barbed congruence. For instance, in the π-calculus, a new channel is guaranteed to remain different from all other existing channels. Thus in a process $\boldsymbol{\nu} a\,(\overline{b}a.\,(a.\, P \mid \overline{c}.\, Q))$, the two prefixes $a.\, P$ and $\overline{c}.\, Q$ may never interact with each other, in any context, even if a is exported. This property does not hold in the Fusion calculus, as a recipient of the newly created name a could equate it with any other name (e.g., using the context $bc.\, \mathbf{0} \mid [\cdot]$).

We do not know whether the encoding of the full π-calculus into $\pi\mathtt{P}$ is fully abstract. However, at least the encoding is fully abstract on the asynchronous subset (where no continuation is allowed after the output prefix).

Theorem 23. *Suppose P, Q are processes from the asynchronous π-calculus, $\mathrm{A}\pi$. Then $P \simeq_{\mathrm{A}\pi} Q$ iff $[\![P]\!] \simeq_{\mathrm{bn}} [\![Q]\!]$.*

In the theorem, $\simeq_{\mathrm{A}\pi}$ could be replaced by $\simeq_{\pi}$ (barbed congruence in the full π-calculus). Note that $\simeq_{\mathrm{A}\pi}$ is the standard barbed congruence, as opposed to *asynchronous* barbed congruence, where output barbs are visible but input barbs are not. We believe the theorem also holds under asynchronous barbed congruence.

For the proof of the theorem, we first establish results of operational correspondence between source and target terms of the encoding. Then the direction from right to left is easy because contexts of the π-calculus are also contexts of $\pi\mathtt{P}$ (under the encoding). The delicate direction is the opposite. Here we use Theorem 21, and the characterisation of π-calculus barbed congruence on the subset of asynchronous processes as ground bisimilarity [5]. We also make use of some up-to techniques, notably 'by-need bisimulation up to $\sim_{\mathrm{bn}}$ and restriction' whose soundness is proved along the lines of soundness proofs of similar techniques for other forms of bisimilarity. We finally consider the relation defined as $\{([\![P]\!] \mid \sigma, [\![Q]\!] \mid \sigma) \;\big|\; P \sim_{\mathrm{g}} Q\}$, where σ is a parallel composition of arcs, and prove that it is a by-need bisimulation up to $\sim_{\mathrm{bn}}$ and up to restriction.

Regarding translations in the opposite direction, both for fusion calculi and for $\pi\mathtt{P}$, the encoding into π is not possible in general. However, for $\pi\mathtt{P}$ some results can be obtained under constraints such as *asynchrony* and *locality*. Something similar has been done by Merro [20] for the Fusion calculus.

VII. Unique negative occurrences of names

In this section we consider a constrained version of the calculi discussed in the paper, where each name may have at most one negative occurrence in a process. In the fusion calculus [6] the constraint means that each name appears at most once as the object of an input. In $\pi\mathtt{P}$, the constraint affects also arcs (as their source is a negative occurrence).

The constraint is rather draconian, bringing the calculi closer to the π-calculus (where the constraint is enforced by having binding input). Still, the constraint is more generous than tying the input to a binder as in π. For instance, we have more complex forms of causality involving input, as in $\boldsymbol{\nu} x(ax.\, \overline{w}t \mid \overline{b}x)$, where the input at a blocks the output at w, and can be triggered before or after the output at b takes place. We call $\pi\mathtt{P1}$ and FU1 the constrained versions of $\pi\mathtt{P}$ and Fusions; in both languages the constraint is preserved by reduction.

We show that the constraint makes certain differences between calculi or semantics disappear. In $\pi\mathtt{P1}$ the eager and the by-need semantics of $\pi\mathtt{P}$ coincide, at least in a weak semantics.

Theorem 24. *In $\pi\mathtt{P1}$, relations $\cong_{\pi\mathtt{P1ea}}$ and $\cong_{\pi\mathtt{P1bn}}$ coincide.*

The following property is useful in the proof (see [23]).

Lemma 25. *For $P \in \pi\mathtt{P1}$, suppose $P \longrightarrow_{\mathrm{ea}} P'$ where the reduction is a rewrite step involving an arc. Then $P \cong_{\pi\mathtt{P1ea}} P'$.*

The calculi $\pi\mathtt{P1}$ and FU1 resulting from the constraint are behaviourally similar. For instance, in $\pi\mathtt{P1}$ the directionality of arcs is irrelevant, as shown by the following law (where we omit the subscripts 'ea' and 'bn' in the light of Theorem 24).

Lemma 26. $a/b \cong_{\pi\mathtt{P1}} b/a$.

Another difference that disappears under the constraint of unique negative occurrences of names is the one concerning capabilities and subtyping in fusion calculi with respect to π and $\pi\mathtt{P}$, exposed in Sections III and IV. Indeed, to equip FU1 with an I/O type system and subtyping, we can use exactly the rules of $\pi\mathtt{P}$ in Section IV-B — with the exception of T-ARC as FU1 does not have arcs. This intuitively because FU1 is, syntactically, a subset of $\pi\mathtt{P}$ (each process of FU1 is also a process of $\pi\mathtt{P}$), and the Subject Reduction theorem for $\pi\mathtt{P}$ in Section IV-B holds regardless of when and how arcs generate substitutions (Remark 11); making an arc a/b act immediately and on all positive occurrences of b is similar to substitution as in FU1. This may however involve changing the type of a name c into a smaller type when c is used in input object; e.g., in $ac \mid (\boldsymbol{\nu} b : T)\overline{a}b.\, P \longrightarrow_{\mathrm{FU1}} P\{c/b\}$ (where $\longrightarrow_{\mathrm{FU1}}$ is reduction in FU1), name c is used at type T, which is a smaller type than $\Gamma(c)$.

Theorem 27. *Let P be a* FU1 *process. If $\Gamma \vdash P$ and $P \longrightarrow_{\mathrm{FU1}} P'$, then $\Gamma' \vdash P'$, where for at most one name c, $\Gamma'(c) \leq \Gamma(c)$; for other names b, $\Gamma'(b) = \Gamma(b)$.*

Note that FU1 does not satisfy the conditions of Definition 2 because well-typed processes may not be freely put in parallel, as this could break the constraint on unique input objects.

We leave for future work a thorough comparison between $\pi\mathtt{P1}$, FU1, and π-calculus.

VIII. FUTURE WORK

Here we mention some lines for future work, in addition to those already mentioned in the main text.

The coinductive characterisation of behavioural equivalence in $\pi\mathtt{P}$ has been presented in the strong case, and should be extended to the weak case. We have presented and compared two semantics for $\pi\mathtt{P}$, eager and by-need. While we tend to consider the advantages so far uncovered for the by-need superior, more work is needed to draw more definite conclusions. For instance, it would also be interesting to contrast axiomatisations of the semantics, rules for pure SOS presentations of the operational semantics, the expressiveness of the subcalculus in which the two semantics agree, and implementations. We do not expect, in contrast, significant differences to arise from type systems.

Another possible advantage of by-need is a smoother extension with dynamic operators like guarded choice, in which an action may discard a component. (In the eager case it is unclear what should be the effect of an arc that acts on one of the summands of a choice.) Choice would be useful for axiomatisations. In by-need, we would have for instance

$$(\boldsymbol{\nu} b, c)\overline{a}b.\, \overline{a}c.\, (\overline{b}|c) \sim (\boldsymbol{\nu} b, c)\overline{a}b.\, \overline{a}c.\, (\overline{b}.\, c + c.\, \overline{b}).$$

The law, valid in both $\pi\mathtt{P}$ and π, illustrates the possibility of generating fresh names that cannot be identified with other names even if exported. The law fails in fusion calculi as a recipient might decide to equate b and c (cf. Section VI-B).

Solos calculus is the polyadic Fusion calculus without continuations. Solos can encode continuations [10]. We believe the same machinery would work for the 'Solos version' of $\pi\mathtt{P}$.

It could also be interesting to study the representation of $\pi\mathtt{P}$ into Psi calculi [21]. This may not be immediate because the latter make use of on an equivalence relation on channels, while the former uses a preorder. One could then see whether the move from Fusions and π to $\pi\mathtt{P}$ in this paper, and the corresponding results on types, can be lifted at the level of Psi calculi, by comparing them with variants based on preorders. [24] presents type systems for Psi calculi, and for explicit fusions, but does not address subtyping.

ACKNOWLEDGEMENT

The authors acknowledge support from the ANR projects 2010-BLAN-0305 PiCoq and 12IS02001 PACE.

REFERENCES

[1] B. Pierce and D. Sangiorgi, "Typing and subtyping for mobile processes," *Math. Str. in Comp. Sci.*, vol. 6, no. 5, pp. 409–453, 1996.

[2] N. Kobayashi, "Type systems for concurrent programs," in *10th Anniversary Colloquium of UNU/IIST*, ser. LNCS, vol. 2757. Springer, 2003, pp. 439–453.

[3] ——, "A new type system for deadlock-free processes," in *CONCUR*, ser. LNCS, vol. 4137. Springer, 2006, pp. 233–247.

[4] K. Honda, V. T. Vasconcelos, , and M. Kubo, "Language primitives and type discipline for structured communication-based programming," in *ESOP*, ser. LNCS, vol. 1381. Springer, 1998, pp. 122—-138.

[5] D. Sangiorgi and D. Walker, *The Pi-Calculus: a theory of mobile processes*. Cambridge University Press, 2001.

[6] J. Parrow and B. Victor, "The fusion calculus: expressiveness and symmetry in mobile processes," in *LICS*. IEEE, 1998, pp. 176 –185.

[7] ——, "The update calculus (extended abstract)," in *AMAST*, ser. LNCS, vol. 1349. Springer, 1997, pp. 409–423.

[8] L. Wischik and P. Gardner, "Explicit fusions," *Theor. Comput. Sci.*, vol. 340, no. 3, pp. 606–630, 2005.

[9] Y. Fu, "The χ-calculus," in *APDC*. IEEE Comp. Soc., 1997, pp. 74–81.

[10] C. Laneve and B. Victor, "Solos in concert," *Math. Str. in Comp. Sci.*, vol. 13, no. 5, pp. 657–683, 2003.

[11] P. Gardner and L. Wischik, "Explicit fusions," in *MFCS*, ser. LNCS, vol. 1893. Springer, 2000, pp. 373–382.

[12] J. Parrow and B. Victor, "The tau-laws of fusion," in *CONCUR*, ser. LNCS, vol. 1466. Springer, 1998, pp. 99–114.

[13] G. L. Ferrari, U. Montanari, E. Tuosto, B. Victor, and K. Yemane, "Modelling Fusion Calculus using HD-Automata," in *CALCO*, ser. LNCS, vol. 3629. Springer, 2005, pp. 142–156.

[14] F. Bonchi, M. G. Buscemi, V. Ciancia, and F. Gadducci, "A presheaf environment for the explicit fusion calculus," *J. Autom. Reasoning*, vol. 49, no. 2, pp. 161–183, 2012.

[15] M. Boreale, M. G. Buscemi, and U. Montanari, "A general name binding mechanism," in *TGC*, ser. LNCS, vol. 3705. Springer, 2005, pp. 61–74.

[16] N. Kobayashi, B. Pierce, and D. Turner, "Linearity and the pi-calculus," *TOPLAS*, vol. 21, no. 5, pp. 914–947, 1999.

[17] K. Honda and N. Yoshida, "On reduction-based process semantics," *Theor. Comp. Sci.*, vol. 152, no. 2, pp. 437–486, 1995.

[18] R. De Nicola and M. Hennessy, "Testing equivalences for processes," *Theor. Comput. Sci.*, vol. 34, pp. 83–133, 1984.

[19] D. Sangiorgi, "Pi-calculus, internal mobility, and agent-passing calculi," *Theor. Comput. Sci.*, vol. 167, no. 1&2, pp. 235–274, 1996.

[20] M. Merro, "Locality in the pi-calculus and applications to distributed objects," Ph.D. dissertation, École des Mines, France, 2000.

[21] J. Bengtson, M. Johansson, J. Parrow, and B. Victor, "Psi-calculi: Mobile processes, nominal data, and logic," in *LICS*. IEEE, 2009, pp. 39—48.

[22] B. Victor, "The fusion calculus : Expressiveness and symmetry in mobile processes," Ph.D. thesis, Uppsala University, 1998.

[23] Web appendix to this paper, available from http://hal.inria.fr/hal-00818068, 2013.

[24] H. Hüttel, "Typed ψ-calculi," in *CONCUR*, ser. LNCS, vol. 6901. Springer, 2011, pp. 265–279.