Symmetries and dualities in name-passing process calculi

Daniel Hirschkoff[1], Jean-Marie Madiot[1], and Davide Sangiorgi[2]

[1] ENS Lyon, Université de Lyon, CNRS, INRIA, France,
[2] INRIA/Università di Bologna, Italy

Abstract. We study symmetries and duality between input and output in the π-calculus. We show that in dualisable versions of π, including πI and fusions, duality breaks with the addition of ordinary input/output types. We illustrate two proposals of calculi that overcome these problems. One approach is based on 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. The second approach consists in taking the minimal symmetrical conservative extension of π with input/output types.

1 Introduction

Process calculi are algebraic models employed for understanding systems of processes: linguistic constructs for concurrency, as well as techniqnes for reasonming about the behaviour of processes. The π-calculus [15] (sometimes simply called π below) is one of the most studied process calculi. In particular, it is the paradigmatical *name-passing* calculus, that is, a calculus where names (a synonymous for "channels") may be passed around. Key aspects for the success of the π-calculus are the minimality of its syntax — its grammar is made of a handful of operators — and its expressiveness — it can model a variety of entities, such as protocols for distributed systems, functions, objects, and so on.

It is common in mathematics to look for symmetries and dualities; dualities may reveal underlying structure and lead to simpler theories. In turn, dualities can be used to relate different mathematical entities. This paper is a summary of work on the π-calculus aimed at studying symmetries and dualities in the π-calculus, particularly those arising in connection with type systems.

In the π-calculus, computation, or reduction, is interaction. This is achieved when an input and an output on the same name meet. If the name is a, then the output, $\overline{a}c.\,P$, emits a name c along a; in the matching input $a(x).\,Q$, name x is bound: it is a placeholder for the object c that is received. The input prefix both sequentialises a behaviour *and* binds a name. Correspondingly, in an interaction two processes are synchronised and, simultaneously, a substitution is performed. The π-calculus features another binder, the restriction operator. These operators, together with parallel composition, are the main operators of the calculus.

Reasoning about processes usually involves proving behavioural equivalences. In the case of the π-calculus, there is a well-established theory of equivalences and proof techniques. In some cases, it is necessary to work in a *typed* setting. Types allow one to express constraints about the observations available to the context when comparing two processes. Indeed, in practice the π-calculus is hardly ever used untyped: a π programmer has always an intended discipline for the use of names in mind; making such discipline explicit by means of types may allow one to validate important behavioural properties which would otherwise fail.

For a simple example, consider a process P that implements two services, for computing the factorial and the exponentiation (n^n) of an integer. The two services are 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 a_1 and a_2:

$$P \stackrel{\text{def}}{=} (\boldsymbol{\nu} a, b)\ (\overline{a_1}\langle a, b\rangle.\, \overline{a_2}\langle a, b\rangle.\, (\mathtt{A} \mid \mathtt{B})) \tag{1}$$

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 a_1 and a_2. The typing rules are straightforward, and mimick those for the typing of references in imperative languages with subtyping. These types, called *capability types*, or i/o-types, are one of the simplest and widely used type disciplines in π.

In the π-calculus, the natural form of duality comes from the symmetry between input and output. There are several variants of π where processes can be 'symmetrised' by replacing inputs with outputs and vice versa. The π-calculus with internal mobility, πI [13], is a subcalculus of π where only bound outputs are allowed (a bound output, that we shall note $\overline{a}(x).\, P$, is the emission of a private name x on some channel a). In πI, duality can be expressed at an operational level, by exchanging (bound) inputs and bound outputs: the dual of $a(x).\, \overline{x}(y).\, 0$ is $\overline{a}(x).\, x(y).\, 0$.

Other well-known variants of π with dualities are the calculi in the fusion family [10, 2, 3]. In fusions, a construct for *free input* acts as the dual of the free output construct of π, and the calculus has only one binder, restriction. Interaction on a given channel has the effect of *fusing* (that is, identifying) names.

As for the π-calculus, however, the amazing expressiveness of the fusion calculi makes desirable behavioural properties fail. The examples we introduced for the π-calculus can be used. For instance, the problems of misbehaving clients of the services (1) remain. Actually, in fusion calculi additional problems arise; for example a client receiving the two channels a and b could fuse them. Now a and b are indistinguishable, and an emission on one of them can reach any of the two services (moreover, if a definition of a service is recursive, a recursive call could be redirected towards the other service).

The i/o-types, while being important for reasoning, bring in some inherent asymmetry. Let us give some intuitions about why it is so. In i/o-types, types are assigned to channels and express *capabilities*: a name of type oT can be used only to emit values of type T, and similarly for the input capability (iT). This is expressed by the following typing rules for i/o-types in π:

$$\frac{\Gamma \vdash a : iT \qquad \Gamma, x : T \vdash P}{\Gamma \vdash a(x).\, P} \qquad\qquad \frac{\Gamma \vdash a : oT \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P}{\Gamma \vdash \overline{a}b.\, P}$$

The rule for input can be read as follows: process $a(x).\, P$ is well-typed provided (i) the typing environment, Γ, ensures that the input capability on a can be derived, and (ii) the continuation of the input can be typed in an environment where x is used according to T. The typing rule for output checks that (i) the output capability on a is derivable, (ii) the emitted value, b, has the right type, and (iii) the continuation P can be typed. As an example, $a : i(iT) \vdash a(x).\, \overline{x}t.\, 0$ cannot be derived, because only the input capability is received on a, which prevents $\overline{x}t.\, 0$ from being typable.

I/o-types come with a notion of subtyping, that makes it possible to relate type $\sharp T$ (which stands for both input and output capabilities) with input and output capabilities (in particular, we have $\sharp T \leq iT$ and $\sharp T \leq oT$). We stress an asymmetry between the constraints attached to the transmitted name in the two rules above. Indeed, while in a reception we somehow enforce a "contract" on the usage of the received name, in the rule for output this is not the case: we can use subtyping in order to derive type, say, iU for b when typechecking the output, while b's type can be $\sharp U$ when typechecking the continuation P.

The main technical point that is discussed in this work is the conflict between the asymmetry inherent to i/o-types and the symmetries we want to obtain via duality. For example i/o-types can be adapted to πI, but duality cannot be applied to the resulting typings. In fusion calculi, the conflict with the asymmetry of i/o-types is even more dramatic. Indeed, subtyping in i/o-types is closely related to substitution, since replacing a name with another makes sense only if the latter has a more general type. Fusions are intuitively substitutions operating in both directions, which leaves no room for subtyping. We explain in Section 3 the problems with symmetries, and why i/o-types cannot be extended easily to fusions.

We discuss two ways to conciliate symmetries and types. The first approach is based on a refinement of fusion calculi. Intuitively, the problems of fusion calculi with types 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 based on a preorder relation (subtyping, set inclusion among subsets of capabilities). The equivalence on names forces one to have an equivalence also on types, instead of a preorder.

The crux of the solution we propose 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. In the processes written above, all visible occurrences of a and c (resp. b) are positive (resp. negative). 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 pro-duced by arcs) only affect the positive occurrences of names.

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.

The second approach to conciliating dualities and types possibility is illustrated by formalising a calculus named $\overline{\pi}$. This is an extension of π with constructs for free input and bound output (note that bound output is not seen as a derived construct in $\overline{\pi}$). In $\overline{\pi}$, we rely on substitutions as the main mechanism at work along interactions. To achieve this, we forbid interactions involving a free input and a free output: the type system rules out processes that use both kinds of prefixes on the same channel.

Calculus $\overline{\pi}$ contains π, and any π process that can be typed using i/o-types can be typed in exactly the same way in $\overline{\pi}$. Moreover, $\overline{\pi}$ contains a 'dualised' version of π: one can choose to use some channels in free input and bound output. For such channels, the typing rules intuitively enforce a 'contract' on the usage of the transmitted name *on the side of the emitter* (dually to the typing rules presented above).

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 [11] and (asymmetric versions of) Chi [2], 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 [10] 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 [1], 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. It is unclear how capability types could be defined in this calculs, as it does not have primitive constructs for input and output.

Structure of the paper . Section 2 gives some background on calculi for mobile processes. Section 3 shows that, in typed languages with fusions, it is impossible to have a non-trivial subtyping, assuming a few simple and standard typing properties of name-passing calculi. Section 4 refines the fusion calculi by replacing the equivalence relation on names generated through communication by a preorder, yielding the calculus $\pi\mathtt{P}$. Finally, Section 5 presents $\overline{\pi}$, the extension of the π-calculus with capability types that enjoys duality properties.

2 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.

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 $a(b).\,P$, free output $\overline{a}b.\,P$, and bout output $\overline{a}(b)$, 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.

The semantics of the calculi studied in the paper are given in the reduction style, by defining structural congruence and reduction relations. 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

3 Typing and subtyping with fusions

Calculi having fusions. When restriction is the only binder (hence the prefixes are 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, Q can be omitted if they are $\mathbf{0}$)

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

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 [11], Chi [2], Fusion [10], Explicit Fusion calculus [3], Solos [8]) have fusions. In Section 4 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 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).

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).[3] The following are the standard typing rules for parallel composition and restriction:

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

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.

[3] 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.

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*.

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

(Prefixes may not have a continuation, in which case P would be missing from the rule.) In the rule, the type of the subject 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, $\Gamma \vdash a : \sharp\, T$ and $\Gamma \vdash b : T$. 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 1. *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 1, a calculus with fusions may only have trivial subtyping.

Theorem 1. *A typed calculus with fusions that is plain and supports narrowing has trivial subtyping.*

In the proof, given in [5], we assume a meaningful subtyping and use it to derive a contradiction from type soundness and the other hypotheses. An additional theorem is presented in [5], showing that any form of narrowing, on one prefix object, would force subtyping to be trivial.

4 A calculus with name preorders

4.1 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 $\pi\mathtt{P}$. 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.

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

We explain the effect of reduction by means of contexts, rather than separate rules for each operator. Contexts yield a more succinct presentation. 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:

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

$$\text{R-Inter}: \quad \overline{a}b.\,P \mid ac.\,Q \longrightarrow P \mid Q \mid b\!\!/\!\!c$$

$$\text{R-SubOut}: \quad a\!\!/\!\!b \mid \overline{b}c.\,Q \longrightarrow a\!\!/\!\!b \mid \overline{a}c.\,Q$$

$$\text{R-SubInp}: \quad a\!\!/\!\!b \mid bc.\,Q \longrightarrow a\!\!/\!\!b \mid ac.\,Q$$

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 $a\!\!/\!\!b$ redirects a prefix at b towards a higher name a. (Arcs remind us of special π-calculus processes, called forwarders or wires [7], 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 $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$. Here are some examples of reduction.

$$\begin{array}{rrl} & \overline{a}c.\overline{c}a.e.P \mid ad.de.\overline{a}.Q & \\ \text{R-Inter} \longrightarrow & \overline{c}a.e.P \mid de.\overline{a}.Q \mid c/d & \\ \text{R-SubInp} \longrightarrow & \overline{c}a.e.P \mid ce.\overline{a}.Q \mid c/d & \\ \text{R-Inter} \longrightarrow & e.P \mid \overline{a}.Q \mid c/d \mid a/e & \\ \text{R-SubInp} \longrightarrow & a.P \mid \overline{a}.Q \mid c/d \mid a/e & \\ \text{R-Inter} \longrightarrow & P \mid Q \mid c/d \mid a/e & \end{array}$$

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:

$$\begin{array}{l} (\boldsymbol{\nu} h, k)\,(bu.\,cu.\,\overline{u} \mid \overline{b}h.\,h.\,P \mid \overline{c}k.\,k.\,Q) \\ \Longrightarrow\ (\boldsymbol{\nu} h, k)\,(\overline{u} \mid h/u \mid k/u \mid h.\,P \mid k.\,Q)\ . \end{array}$$

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:

$$\begin{array}{r} \text{R-SubOut} \longrightarrow (\boldsymbol{\nu} h, k)\,(\overline{k} \mid h/u \mid k/u \mid h.\,P \mid k.\,Q) \\ \text{R-Inter} \longrightarrow (\boldsymbol{\nu} h, k)\,(h/u \mid k/u \mid h.\,P \mid Q)\ . \end{array}$$

Definition 2 (Positive and negative occurrences). *In an input $ab.\,P$ and an arc a/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 [5]).

4.2 Types

We now show that the i/o capability type system and its subtyping can be transplanted from π to $\pi\mathtt{P}$.

In the typing rules for i/o-types in the (monadic) π-calculus [12], 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. The subsumption rule injects subtyping in the typing rules. The most important typing rules are those for input and output prefixes; for input we have:

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

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

Subtyping rules:

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

$$\frac{S \leq T \qquad T \leq U}{S \leq U}$$

Typing rules:

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

$$\textsc{T-Nil}\ \frac{}{\Gamma \vdash \mathbf{0}} \qquad \textsc{T-Out}\ \frac{\Gamma \vdash a : \mathtt{o}\,T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P}{\Gamma \vdash \overline{a} b.\, P} \qquad \textsc{T-InpFree}\ \frac{\Gamma \vdash a : \mathtt{i}\,\Gamma(b) \qquad \Gamma \vdash P}{\Gamma \vdash a b.\, P}$$

$$\textsc{T-Arc}\ \frac{\Gamma \vdash a : \Gamma(b)}{\Gamma \vdash a/b}$$

Table 1. The type system of $\pi\mathtt{P}$

The type system for $\pi\mathtt{P}$ is presented in Table 1. 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/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 the premises of 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 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, $P \longrightarrow c\not b \mid \overline{b} \longrightarrow c\not b \mid \overline{c}$, 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 2 (Polarised narrowing). *Let T_1, T_2 be 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 2 (specialised to prefixes) does not contradict Theorem 1, because in $\pi\mathtt{P}$, reduction does not satisfy (2) (from Section 2). We have subject reduction:

Theorem 3. *If $\Gamma \vdash P$ and $P \longrightarrow P'$ then also $\Gamma \vdash P'$.*

4.3 Other results

Behavioural equivalences for $\pi\mathtt{P}$ and the fusion calculi, in the form of barbed congruence, are considered in [5]. It is shown that the modification from fusion calculi to $\pi\mathtt{P}$ 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. Indeed $\pi\mathtt{P}$ is closer to the π-calculus than to fusion calculi, not only in typing, but also behaviourally.

The reduction semantics for $\pi\mathtt{P}$ that we have presented ma considered *eager*, in that arcs may freely act on prefixes. An alternative, *by-need*, semantics, is possible, where arcs act on prefixes only when interactions occur. See [5] for a comparison between the two semantics, as well as for further comparison with semantics based on name fusion. The behavioural theory of $\pi\mathtt{P}$, under by-need semantics, is further studied in [6]. Two characterisations of barbed congruence, using a labelled transition system and using equations, are presented. Also, see [5] for examples concerning behavioural laws and expressiveness results for $\pi\mathtt{P}$.

5 $\overline{\pi}$, a symmetric π-calculus

In this section, we present $\overline{\pi}$, a π-calculus with i/o-types that enjoys duality properties. We define the syntax and operational semantics for $\overline{\pi}$ processes in Section 5.1, introduce types and barbed congruence in Section 5.2, establish duality in Section 5.3. We finally discuss other results, and an application to the encoding of functions, in Section 5.4.

5.1 Syntax and Operational Semantics

The syntax of $\overline{\pi}$ is as follows:

$$P ::= 0 \;\big|\; P \mid P \;\big|\; \alpha.\, P \;\big|\; (\nu a)P \qquad \alpha ::= \rho b \mid \rho(x) \qquad \rho ::= a \mid \overline{a}$$

$\overline{\pi}$ differs from the usual π-calculus by the presence of the free input ab and bound output $\overline{a}(x)$ prefixes. Note that in $\overline{\pi}$, the latter is *not* a notation for $(\nu x)\overline{a}x.\, P$, but a primitive construct. These prefixes are the symmetric counterpart of $\overline{a}b$ and $a(x)$ respectively. Given ρ of the form a or $\overline{a}$, $\mathrm{n}(\rho)$ is defined by

Reduction is defined by law R-SCon from Section 4.1, as well as the following axioms, to allow communication involving two prefixes *only if at least one of them is bound*:

$$\overline{a}b.\, P \mid a(x).\, Q \;\to P \mid Q[b/x] \qquad ab.\, P \mid \overline{a}(x).\, Q \;\to P \mid Q[b/x]$$

$$\overline{a}(x).\, P \mid a(x).\, Q \;\to (\nu x)(P \mid Q)$$

$\Rightarrow$ denotes the reflexive transitive closure of $\to$. Note that the $\overline{\pi}$ process $\overline{a}b \mid ab$ has no reduction; this process is ruled out by the type system presented below.

5.2 Types and Behavioural Equivalence

Types in $\overline{\pi}$ are a refinement of standard i/o-types: in addition to capabilities (ranged over using c), we annotate types with *sorts* (s), that specify whether a name can be used in free input (sort $\mathbf{e}$) or in free output ($\mathbf{r}$) — note that a name cannot be used to build both kinds of free prefixes.

$$T ::= c^s T \mid \mathbf{1} \qquad c ::= i \mid o \mid \sharp \qquad s ::= \mathbf{e} \mid \mathbf{r}$$

If name a has type $c^{\mathbf{r}}T$, we shall refer to a as a $\mathbf{r}$-*name*, and similarly for $\mathbf{e}$.

The subtyping relation is the smallest reflexive and transitive relation $\leq$ satisfying the rules of Figure 1. As in the π-calculus $i^{\mathbf{r}}$ is covariant and $o^{\mathbf{r}}$ is contravariant. Dually, $i^{\mathbf{e}}$ is contravariant and $o^{\mathbf{e}}$ is covariant. Note that sorts ($\mathbf{e}$, $\mathbf{r}$) are not affected by subtyping.

The type system is given by the rules of Figure 2. We write $\Gamma(a)$ for the type associated to a in Γ. There is a dedicated typing rule for every kind of prefix (free, ρb, or bound, $\rho(x)$), according to the sort of the involved name. $T^{\leftrightarrow}$ stands for T where we switch the toplevel capability: $(c^s T)^{\leftrightarrow} = \overline{c}^s T$ where

$$\frac{}{\sharp^s T \leq i^s T} \qquad \frac{}{\sharp^s T \leq o^s T}$$

$$\frac{T_1 \leq T_2}{i^{\mathbf{r}} T_1 \leq i^{\mathbf{r}} T_2} \qquad \frac{T_1 \leq T_2}{o^{\mathbf{r}} T_2 \leq o^{\mathbf{r}} T_1} \qquad \frac{T_1 \leq T_2}{i^{\mathbf{e}} T_2 \leq i^{\mathbf{e}} T_1} \qquad \frac{T_1 \leq T_2}{o^{\mathbf{e}} T_1 \leq o^{\mathbf{e}} T_2}$$

Fig. 1. Subtyping in $\overline{\pi}$

$$\frac{\Gamma \vdash a : i^{\mathbf{r}} T \qquad \Gamma, x : T \vdash P}{\Gamma \vdash a(x).\, P} \qquad \frac{\Gamma \vdash a : i^{\mathbf{e}} T \qquad \Gamma, x : T^{\leftrightarrow} \vdash P}{\Gamma \vdash a(x).\, P}$$

$$\frac{\Gamma \vdash a : o^{\mathbf{e}} T \qquad \Gamma, x : T \vdash P}{\Gamma \vdash \overline{a}(x).\, P} \qquad \frac{\Gamma \vdash a : o^{\mathbf{r}} T \qquad \Gamma, x : T^{\leftrightarrow} \vdash P}{\Gamma \vdash \overline{a}(x).\, P}$$

$$\frac{\Gamma \vdash a : i^{\mathbf{e}} T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P}{\Gamma \vdash ab.\, P} \qquad \frac{\Gamma \vdash a : o^{\mathbf{r}} T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P}{\Gamma \vdash \overline{a}b.\, P}$$

$$\frac{\Gamma, a : T \vdash P}{\Gamma \vdash (\nu a) P} \qquad \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad \frac{}{\Gamma \vdash 0} \qquad \frac{\Gamma(a) \leq T}{\Gamma \vdash a : T}$$

Fig. 2. $\overline{\pi}$: Typing rules

$\overline{o} = i, \overline{i} = o, \overline{\sharp} = \sharp$. The typing rules for **r**-names impose a constraint on the **r**eceiving side: all inputs on a **r**-channel should be bound. Note that $\overline{a}(x).\, P$ and $(\boldsymbol{\nu} x)\overline{a}x.\, P$ are *not* equivalent from the point of view of typing: typing a bound output on a **r**-channel (a) imposes that the transmitted name (x) is used according to the "dual constraint" w.r.t. what a's type specifies: this is enforced using $T^{\leftrightarrow}$ (while names received on a are used according to T). Symmetrical considerations hold for **e**-names, that impose constraints on the **e**mitting side.

Remark 1 ("Double contract"). We could adopt a more liberal typing for bound outputs on **r** names, and use the rule

$$\frac{\Gamma \vdash a : o^{\mathbf{r}} T \qquad \Gamma, x : T' \vdash P \qquad T' \leq T}{\Gamma \vdash \overline{a}(x).\, P}$$

(and its counterpart for inputs on **e**-names). This would have the effect of typing $\overline{a}(x).\, P$ like $(\boldsymbol{\nu} x)\overline{a}x.\, P$. We instead chose to enforce what we call a *"double contract"*: the same way a receiving process uses the bound name according to the type specified in the channel that is used for reception, the continuation of a bound output uses the emitted name according to $T^{\leftrightarrow}$, the *symmetrised version* of T. This corresponds to a useful programming idiom in π, where it is common to create a name, transmit one capability on this name and use locally the other, dual capability. This choice moreover simplifies reasoning about $\overline{\pi}$.

Observe that when a typable process reduces according to

$$\overline{a}(x).\, P \mid a(x).\, Q \to (\nu x)(P \mid Q) \ ,$$

if a has type, say, $\sharp^{\mathbf{r}}(o^s T)$, then in the right hand side process, name x is given type $\sharp^s T$, and the $\sharp$ capability is "split" into $i^s T$ (used by P) and $o^s T$ (used by Q). It would be the other way around if a's sort were $\mathbf{e}$.

Proposition 1 (Subject reduction). *If $\Gamma \vdash P$ and $P \to Q$ then $\Gamma \vdash Q$.*

We now move to the definition of behavioural equivalence.

Definition 3 (Contexts). *Contexts are processes with one occurrence of the hole, written $[-]$. They are defined by the following grammar:*

$$C ::= [-] \mid C \mid P \mid C \mid P \mid \alpha.\, C \mid (\nu a)C \ .$$

Definition 4. *Let Γ, Δ be typing environments. We say that Γ* extends *Δ if the support of Δ is included in the support of Γ, and if $\Delta \vdash x : T$ entails $\Gamma \vdash x : T$ for all x. A context C is a (Γ/Δ)-context, written $\Gamma/\Delta \vdash C$, if C can be typed in the environment Γ, the hole being well-typed in any context that extends Δ.*

Definition 5 (Barbs). *Given $\rho \in \{a, \overline{a}\}$, where a is a name, we say that P exhibits barb ρ, written $P{\downarrow_\rho}$, if $P \equiv (\nu c_1 \ldots c_n)(\alpha.\, Q \mid R)$ where $\alpha \in \{\rho(x), \rho b\}$ with $a \notin \{c_1, \ldots, c_n\}$. We extend the definition to* weak barbs*: $P \Downarrow_\rho$ stands for $P \Rightarrow{\downarrow_\rho}$.*

Definition 6 (Typed barbed congruence). Barbed bisimilarity *is the largest symmetric relation $\dot{\approx}$ such that whenever $P \dot{\approx} Q$,*

1. *if $P \downarrow_\rho$ then $Q \Downarrow_\rho$, and*
2. *if $P \to P'$ then $Q \Rightarrow \dot{\approx} P'$.*

When $\Delta \vdash P$ and $\Delta \vdash Q$, we say that P and Q are barbed congruent at Δ, *written $\Delta \rhd P \cong^c Q$, if for all (Γ/Δ)-context C, $C[P] \dot{\approx} C[Q]$.*

5.3 Duality

Definition 7 (Dual of a process). *The dual of a process P, written $\overline{P}$, is the process obtained by transforming prefixes as follows: $\overline{\overline{a}b} = ab$, $\overline{ab} = \overline{a}b$, $\overline{\overline{a}(x)} = a(x)$, $\overline{a(x)} = \overline{a}(x)$, and applying dualisation homeomorphically to the other constructs.*

Lemma 1 (Duality for reduction). *If $P \to Q$ then $\overline{P} \to \overline{Q}$.*

Dualising a type means swapping i/o capabilities and $\mathbf{e}/\mathbf{r}$ sorts.

Definition 8 (Dual of a type). *The dual of T, written $\overline{T}$, is defined as follows:*

$$\overline{c^s T} = \overline{c}^{\overline{s}}\, \overline{T} \qquad \text{with} \qquad \overline{\mathbf{r}} = \mathbf{e}, \quad \overline{\mathbf{e}} = \mathbf{r}, \quad \overline{i} = o, \quad \overline{o} = i \ .$$

We extend the definition to typing environments, and write $\overline{\Gamma}$ for the dual of Γ.

Lemma 2 (Duality for typing). *The type system enjoys the following duality properties: If* $T_1 \leq T_2$ *then* $\overline{T_1} \leq \overline{T_2}$*. Moreover, if* $\Gamma \vdash P$ *then* $\overline{\Gamma} \vdash \overline{P}$*. Finally, if* $\Gamma/\Delta \vdash C$ *then* $\overline{\Gamma}/\overline{\Delta} \vdash \overline{C}$*.*

Most importantly, duality holds for typed barbed congruence. The result is easy in the untyped case, since duality preserves reduction and dualises barbs. On the other hand, we are not aware of the existence of another system having this property in presence of i/o-types.

Theorem 4 (Duality for $\cong^c$). *If* $\Delta \rhd P \cong^c Q$ *then* $\overline{\Delta} \rhd \overline{P} \cong^c \overline{Q}$*.*

5.4 Further results and applications

It is shown in [4] that $\overline{\pi}$ can be related to π, by translating $\overline{\pi}$ into a variant of the π-calculus with i/o-types in a a fully abstract way. This result shows that π and $\overline{\pi}$ are rather close in terms of expressiveness.

As an application of $\overline{\pi}$, its dualities, and its behavioural theory, the calculus is used in [4] to relate two encodings of call-by-name λ-calculus. The first one is the ordinary encoding by Milner [9], the second one, more recent, is by van Bakel and Vigliotti [16]. The two encodings are syntactically quite different. Milner's is *input-based*, in that an abstraction interacts with its environment via an input. In contrast, van Bakel and Vigliotti's is *output-based*.

We exploit $\overline{\pi}$ (in fact the extension of $\overline{\pi}$ with delayed input) to prove that the two encodings are the dual of one another. This is achieved by first embedding the π-terms of the λ-encodings into $\overline{\pi}$, and then applying behavioural laws of $\overline{\pi}$. The correctness of these transformations is justified using i/o-types (essentially to express the conditions under which a link can be erased in favour of a substitution). As a consequence, correctness results for one encoding can be transferred onto the other one. For instance, we derive that the equivalence induced on λ-terms by Milner's encoding (whereby two λ-terms are equal if their π-calculus images are behaviourally equivalent) is the same as that induced by van Bakel and Vigliotti's encoding. And since for Milner's encoding this equivalence coincides with the Levy-Longo tree equality [14], the same holds for van Bakel and Vigliotti's encoding, a question that is not addressed in [16].

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

The third author, Davide Sangiorgi, would like to add special thanks to Jozef Gruska: my current research is quite remote from the systolic automata and systems on which I collaborated with Jozef when I was a student; I have learned a lot from Jozef's papers and from our collaboration, and it is a pleasure to contribute a paper in this volume.

References

1. M. Boreale, M. G. Buscemi, and U. Montanari. A General Name Binding Mechanism. In *TGC*, volume 3705 of *LNCS*, pages 61–74. Springer, 2005.

2. Y. Fu. The χ-calculus. In *Proc. APDC*, pages 74–81. IEEE Computer Society Press, 1997.
3. P. Gardner and L. Wischik. Explicit fusions. In *Proc. of MFCS*, volume 1893 of *LNCS*, pages 373–382. Springer-Verlag, 2000.
4. D. Hirschkoff, J.-M. Madiot, and D. Sangiorgi. Duality and i/o-types in the π-calculus. In *Proc. of CONCUR*, volume 7454 of *LNCS*, pages 302–316. Springer, 2012.
5. D. Hirschkoff, J.-M. Madiot, and D. Sangiorgi. Name-Passing Calculi: From Fusions to Preorders and Types. long version of the paper presented at LICS'13, in preparation, 2014.
6. D. Hirschkoff, J.-M. Madiot, and X. Xu. A behavioural theory for a π-calculus with preorders. submitted, 2014.
7. K. Honda and N. Yoshida. On reduction-based process semantics. *Theor. Comp. Sci.*, 152(2):437–486, 1995.
8. C. Laneve and B. Victor. Solos in Concert. *Mathematical Structures in Computer Science*, 13(5):657–683, 2003.
9. R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.
10. J. Parrow and B. Victor. The fusion calculus: expressiveness and symmetry in mobile processes. In *Proc. of LICS*, pages 176 –185. IEEE, 1998.
11. Joachim Parrow and Björn Victor. The update calculus (extended abstract). In *AMAST*, volume 1349 of *LNCS*, pages 409–423. Springer, 1997.
12. B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.
13. D. Sangiorgi. π-calculus, internal mobility, and agent-passing calculi. In *Selected papers from TAPSOFT '95*, pages 235–274. Elsevier, 1996.
14. D. Sangiorgi. Lazy functions and mobile processes. In *Proof, Language, and Interaction*, pages 691–720. The MIT Press, 2000.
15. D. Sangiorgi and D. Walker. *The Pi-Calculus: a theory of mobile processes*. Cambridge University Press, 2001.
16. S. van Bakel and M. G. Vigliotti. An Implicative Logic based encoding of the λ-calculus into the π-calculus, 2014. From `http://www.doc.ic.ac.uk/~svb/`.