Duality and i/o-types in the π-calculus

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 duality between input and output in the π-calculus. In dualisable versions of π, including πI and fusions, duality breaks with the addition of ordinary input/output types. We introduce $\overline{\pi}$, intuitively the minimal symmetrical conservative extension of π with input/output types. We prove some duality properties for $\overline{\pi}$ and we study embeddings between $\overline{\pi}$ and π in both directions. As an example of application of the dualities, we exploit the dualities of $\overline{\pi}$ and its theory to relate two encodings of call-by-name λ-calculus, by Milner and by van Bakel and Vigliotti, syntactically quite different from each other.

1 Introduction

It is common in mathematics to look for dualities; dualities may reveal underlying structure and lead to simpler theories. In turn, dualities can be used to relate different mathematical entities. In this work, our goal is to study dualities in the typed π-calculus, and to exploit them to understand the possible relationships between encodings of functions as π-calculus processes.

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. One of the simplest and widely used such discipline is given by input/output-types [SW01] — i/o-types in the sequel.

In the π-calculus (simply called π below), 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 [San96], 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 [PV98,Fu97,GW00]. 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.

The discipline of simple types can be adapted both to πI and to fusions, while preserving dualities. The situation is less clear for i/o-types, which can

be very useful to establish equivalences between processes. 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 starting point of 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. In work in preparation [HMS12], we investigate this relationship between subtyping and substitution, and compare several variants of existing calculi, including the one presented in this paper.

In this paper, in order to work in a setting that provides a form of duality and where i/o-types can be used, we introduce a calculus named $\overline{\pi}$ (Section 2). $\overline{\pi}$ 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). We show how $\overline{\pi}$ can be related to π, by translating $\overline{\pi}$ into a variant of the π-calculus with i/o-types in a fully abstract way. This result shows that π and $\overline{\pi}$ are rather close in terms of expressiveness.

We also define a notion of typed barbed congruence in $\overline{\pi}$, which allows us to validate at a behavioural level the properties we have mentioned above: two processes are equivalent if and only if their duals are. To our knowledge, no existing calculus with i/o-types enjoys this form of duality for behaviours.

As an application of $\overline{\pi}$, its dualities, and its behavioural theory, we use $\overline{\pi}$ to relate two encodings of call-by-name λ-calculus. The first one is the ordinary encoding by Milner [Mil92], the second one is by van Bakel and Vigliotti [vBV09]. 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*. Moreover, only the latter makes use of *link processes*, that is, forwarders that under certain conditions act as substitutions.

Van Bakel and Vigliotti actually encode *strong* call-by-name — reductions may also take place inside a λ-abstraction. We therefore compare van Bakel and Vigliotti's encoding with the strong variant of Milner's encoding, obtained by replacing an input with a delayed input, following [Mer00] (in a delayed input $a(x){:}P$, the continuation P may perform transitions not involving the binder x even when the head input at a has not been consumed).

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). Some of the transformations needed for the λ-encodings, however, are proved in this paper only for barbed bisimilarity; see the concluding section for a discussion.

Paper outline. Section 2 introduces $\overline{\pi}$, and presents its main properties. To analyse dualities in encodings of λ into π, in Section 3, we extend $\overline{\pi}$, notably with delayed prefixes. As the addition of these constructs is standard, they are omitted from the original syntax so to simplify the presentation. Section 4 gives concluding remarks.

2 $\overline{\pi}$, 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 2.1, introduce types and barbed congruence in Section 2.2, establish duality in Section 2.3, and present results relating π and $\overline{\pi}$ in Section 2.4.

2.1 Syntax and Operational Semantics

We consider an infinite set of names, ranged over using $a, b, \ldots, x, y, \ldots$. The syntax of $\overline{\pi}$ is as follows:

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

$\overline{\pi}$ 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 a process P, $\mathrm{fn}(P)$ stands for the set of free names of P — restriction, bound input and bound output are binding constructs. Given ρ of the form a or $\overline{a}$, $\mathrm{n}(\rho)$ is defined by $\mathrm{n}(\overline{a}) = \mathrm{n}(a) = a$.

Structural congruence is standard, and defined as in π (in particular, there are no axioms involving prefixes). The reduction laws allow communication involving two prefixes *only if at least one of them is bound*:

$$\begin{array}{rr}
\overline{a}b.P \mid a(x).Q \to P \mid Q[b/x] & P \to Q \text{ if } P \equiv \to \equiv Q \\
ab.P \mid \overline{a}(x).Q \to P \mid Q[b/x] & (\nu a)P \to (\nu a)Q \text{ if } P \to Q \\
\overline{a}(x).P \mid a(x).Q \to (\nu x)(P \mid Q) & P \mid R \to Q \mid R \text{ if } P \to Q
\end{array}$$

Note that $\overline{a}b \mid ac$ is a process of $\overline{\pi}$ that has no reduction; this process is ruled out by the type system presented below.

2.2 Types and Behavioural Equivalence

Types 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 an $\mathbf{r}$-*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 defined as a refinement of input/output types, and is given by the rules of Figure 2. 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. We write $\Gamma(a)$ for the type associated to a in Γ. $T^{\leftrightarrow}$ stands for T where we switch the top-level capability: $(c^s T)^{\leftrightarrow} = \overline{c}^s T$ where $\overline{o} = i, \overline{i} = o, \overline{\sharp} = \sharp$. The typing rules for $\mathbf{r}$-names impose a constraint on the $\mathbf{r}$eceiving side: all inputs on an $\mathbf{r}$-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 an $\mathbf{r}$-channel (a) imposes that the transmitted name (x) is used according to the "dual constraint" w.r.t.

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

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

Fig. 1. Subtyping

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

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

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

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

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

what a's type specifies: this is enforced using $T^{\leftrightarrow}$ (while names received on a are used according to T). Symmetrical considerations can be made for **e**-names, that impose constraints on the **e**mitting side.

We write $\Gamma \vdash P, Q$ when both $\Gamma \vdash P$ and $\Gamma \vdash Q$ can be derived.

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 idiom is used e.g. in [Vas09] and in [SW01, Sect. 5.7.3]. This choice moreover makes the proofs in Section 3.2 easier.

Observe that when a typable process reduces according to

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

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 $\mathbf{e}$.

Lemma 1 (Properties of typing).

1. *(Weakening) If* $\Gamma \vdash P$ *then* $\Gamma, a : T \vdash P$.
2. *(Strengthening) If* $\Gamma, a : T \vdash P$ *and* $a \notin fn(P)$ *then* $\Gamma \vdash P$.
3. *(Narrowing) If* $\Delta \leq \Gamma$ *and* $\Gamma \vdash P$ *then* $\Delta \vdash P$.
4. *(Substitution) If* $\Gamma, x : T \vdash P$ *and* $\Gamma \vdash b : T$ *then* $\Gamma \vdash P[b/x]$.

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

Proof. By transition induction. Lemma 1 (4) is used when a bound prefix communicates with a free prefix; Lemma 1 (3) is used for the interaction between two bound prefixes, since T and $T^{\leftrightarrow}$ have a common subtype. □

Definition 1 (Contexts). *Contexts are processes with one occurrence of the hole, written* $[-]$. *They are defined by the following grammar:*

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

Definition 2. *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.

As a consequence of the previous definition and of Lemma 1, it is easy to show that if $\Delta \vdash P$ and $\Gamma/\Delta \vdash C$, then $\Gamma \vdash C[P]$.

We now move to the definition of behavioural equivalence.

Definition 3 (Barbs). *Given* $\rho \in \{a, \overline{a}\}$, *where* a *is a name, we say that* P *exhibits barb* ρ, *written* $P{\downarrow_\rho}$, *if* $P \equiv (\nu c_1 \dots c_n)(\alpha.Q \mid R)$ *where* $\alpha \in \{\rho(x), \rho b\}$ *with* $a \notin \{c_1, \dots, c_n\}$. *We extend the definition to* weak barbs*:* $P \Downarrow_\rho$ *stands for* $P \Rightarrow{\downarrow_\rho}$ *where* $\Rightarrow$ *is the reflexive transitive closure of* $\to$.

Definition 4 (Typed barbed congruence). Barbed bisimilarity *is the largest symmetric relation* $\dot{\approx}$ *such that whenever* $P \dot{\approx} Q$, $P \downarrow_\rho$ *implies* $Q \Downarrow_\rho$ *and* $P \to P'$ *implies* $Q \Rightarrow \dot{\approx} P'$. *When* $\Delta \vdash P, Q$, *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, $C[P] \dot{\approx} C[Q]$.

2.3 Duality

Definition 5 (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 2 (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 6 (Dual of a type). *The dual of T, written $\overline{T}$, is defined by setting $\overline{c^s T} = \overline{c}^{\overline{s}}\,\overline{T}$, with $\overline{\mathbf{r}} = \mathbf{e}, \overline{\mathbf{e}} = \mathbf{r}, \overline{i} = o$, and $\overline{o} = i$. We extend the definition to typing environments, and write $\overline{\Gamma}$ for the dual of Γ.*

Lemma 3 (Duality for typing).

1. *If $T_1 \leq T_2$ then $\overline{T_1} \leq \overline{T_2}$.*
2. *If $\Gamma \vdash P$ then $\overline{\Gamma} \vdash \overline{P}$.*
3. *If $\Gamma/\Delta \vdash C$ then $\overline{\Gamma}/\overline{\Delta} \vdash \overline{C}$.*

Proof. (1): the covariant type operators ($i^{\mathbf{r}}$ and $o^{\mathbf{e}}$) are dual of each other, and so are the contravariant operators ($o^{\mathbf{r}}$ and $i^{\mathbf{e}}$). (2) follows from the shape of the typing rules, e.g., the dual of the rule for $i^{\mathbf{r}}$ is an instance of the rule for $\overline{i^{\mathbf{r}}} = o^{\mathbf{e}}$. (3) holds because if Φ extends Δ then $\overline{\Phi}$ extends $\overline{\Delta}$ (item (1)). □

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

Proof. By Lemma 3, we only have to prove that if $P \mathrel{\dot{\approx}} Q$ then $\overline{P} \mathrel{\dot{\approx}} \overline{Q}$, i.e., duality preserves reduction and swaps barbs. □

2.4 Embeddings between π and $\overline{\pi}$

From $\overline{\pi}$ to π^{io}. As explained in Section 1, the π-calculus with i/o-types (that we note π^{io}) is an asymmetric calculus. In some sense, $\overline{\pi}$ can be seen as a 'dualisation' of π^{io}. This can be formulated rigorously by projecting $\overline{\pi}$ into π^{io}. To define this projection, which we call a *partial dualisation*, we work in an extended version of π^{io}, where capabilities are duplicated: in addition to the $i, o, \sharp$ capabilities, we also have capabilities $\underline{i}, \underline{o}$ and $\underline{\sharp}$, that intuitively correspond to the image of the "$\mathbf{e}$-part" of $\overline{\pi}$ through the encoding. The additional capabilities act exactly like the corresponding usual capabilities, in particular w.r.t. subtyping and duality. We write π_2^{io} for the resulting calculus. We discuss below (Remark 3) to what extent the addition of these capabilities is necessary. We also rely on π_2^{io} to prove that $\overline{\pi}$ is a conservative extension of the π-calculus in Theorem 2 — π_2^{io} is actually close, operationally, to both calculi.

Definition 7 (Partial dualisation). *We define a translation from typed processes in $\overline{\pi}$ to π_2^{io}. The translation acts on typing derivations: given a derivation δ of $\Gamma \vdash P$ (written $\delta :: \Gamma_\delta \vdash P$), we define a π_2^{io} process noted $[P]^\delta$ as follows:*

$$
\begin{array}{rl}
[\rho b.P]^\delta = \overline{\rho} b.[P]^{\delta'} & \text{if } \Gamma_\delta(\mathrm{n}(\rho)) = c^{\mathbf{e}}T \\
[\rho b.P]^\delta = \rho b.[P]^{\delta'} & \text{if } \Gamma_\delta(\mathrm{n}(\rho)) = c^{\mathbf{r}}T \\
[\rho(x).P]^\delta = \overline{\rho}(x).[P]^{\delta'} & \text{if } \Gamma_\delta(\mathrm{n}(\rho)) = c^{\mathbf{e}}T \\
[\rho(x).P]^\delta = \rho(x).[P]^{\delta'} & \text{if } \Gamma_\delta(\mathrm{n}(\rho)) = c^{\mathbf{r}}T
\end{array}
$$

$$
[(\nu a)P]^\delta = [P]^{\delta'} \qquad [0]^\delta = 0 \qquad [!P]^\delta = ![P]^{\delta'} \qquad [P \mid Q]^\delta = [P]^{\delta'_1} \mid [Q]^{\delta'_2}
$$

In the above definition, δ' is the subderivation of δ, in case there is only one, and δ'_1 and δ'_2 are the obvious subderivations in the case of parallel composition. We extend the definition to types: T^* stands for T where all occurrences of $c^{\mathbf{r}}$ (resp. $c^{\mathbf{e}}$) are replaced with c (resp. $\overline{c}$, the dual of $\underline{c}$). We define accordingly Γ^*.

Remark 2. The same translation could be defined for a simply typed version of $\overline{\pi}$. Indeed, $[\cdot]^{\text{-}}$ does not depend on capabilities ($i/o/\sharp$), but only on sorts ($\mathbf{r}/\mathbf{e}$).

Lemma 4. *If $\delta :: \Gamma \vdash P$ (in $\overline{\pi}$), then $\Gamma^* \vdash [P]^\delta$ (in π_2^{io}).*

Proof. In moving from Γ to Γ^*, we replace $i^{\mathbf{e}}$ (resp. $o^{\mathbf{e}}$, $i^{\mathbf{r}}$, $o^{\mathbf{r}}$) with $\underline{o}$ (resp. $\underline{i}$, i, o). This transformation preserves the subtyping relation. Moreover, the rules to type prefixes $i^{\mathbf{r}}, o^{\mathbf{r}}, i^{\mathbf{e}}, o^{\mathbf{e}}$ in $\overline{\pi}$ correspond to the rules for $i, o, \underline{o}, \underline{i}$ in π_2^{io}. □

Lemma 5. *Whenever $\delta_1 :: \Gamma \vdash P$ and $\delta_2 :: \Gamma \vdash P$, we have $\Gamma^* \rhd [P]^{\delta_1} \simeq^c [P]^{\delta_2}$.*

Proof. The relation $\mathcal{R} \triangleq \{([P]^{\delta_1}, [P]^{\delta_2}) \mid \delta_1, \delta_2 :: \Gamma \vdash P\}$ is a strong bisimulation in π and is substitution-closed; hence $\mathcal{R}$ is included in $\simeq^c$, since $[P]^{\delta_i}$ is typable in Γ^* (by Lemma 4). □

Lemma 6. *If $\delta_P :: \Gamma \vdash P$ and $\delta_Q :: \Gamma \vdash Q$ then we have the following:*

1. *(P and Q have the same barbs) iff ($[P]^{\delta_P}$ and $[Q]^{\delta_Q}$ have the same barbs)*
2. *if $P \to P'$ then $[P]^{\delta_P} \to [P']^{\delta}$ for some $\delta :: \Gamma \vdash P'$.*
3. *if $[P]^{\delta_P} \to P_1$ then $P_1 = [P']^{\delta}$ with $P \to P'$ for some $\delta :: \Gamma \vdash P'$.*
4. *$P \dot{\approx} Q$ iff $[P]^{\delta_P} \dot{\approx} [Q]^{\delta_Q}$.*

Proof. (4) is a consequence of (1), (2), (3). For (1) remark that if $\Gamma(a) = c^{\mathbf{r}}T$ then P and $[P]^{\delta_P}$ have the same barbs on a; if $\Gamma(a) = c^{\mathbf{e}}T$, they have dual barbs on a, but in this case so do Q and $[Q]^{\delta_Q}$. For (2) and (3), we remark that $[\cdot]^\delta$ is compositional and preserves the fact that two prefixes can interact — even when moving to a different δ. □

Proposition 2 (Full abstraction). *If $\delta_P :: \Gamma \vdash P$ and $\delta_Q :: \Gamma \vdash Q$ then*

$$\Gamma \rhd P \cong^c Q \text{ (in } \overline{\pi}) \quad \text{iff} \quad \Gamma^* \rhd [P]^{\delta_P} \cong^c [Q]^{\delta_Q} \text{ (in } \pi_2^{\text{io}}) \ .$$

Proof. Soundness: given a derivation $\gamma :: \Delta/\Gamma \vdash C$, we build $[C]^\gamma$ which is a (Δ^*/Γ^*)-context. Then $[C]^\gamma[[P]^{\delta_P}] = [C[P]]^{\beta_P}$ for some β_P and we can rely on barbed congruence in π_2^{io} to establish $[C[P]]^{\beta_P} \dot{\approx} [C[Q]]^{\beta_Q}$. By Lemma 6, we deduce $C[P] \dot{\approx} C[Q]$.
Completeness: we define the reverse translation $\{\cdot\}^{\text{-}}$ of $[\cdot]^{\text{-}}$ and reason as above to prove its soundness. Thanks to the fact that $\delta_P :: \Gamma \vdash P$ implies $\{[P]^{\delta_P}\}^{\delta_P^*} = P$ where $\delta_P^* :: \Gamma^* \vdash [P]^\delta$ is the derivation obtained by Lemma 4, the soundness of $\{\cdot\}^{\text{-}}$ implies the completeness of $[\cdot]^{\text{-}}$, and vice versa. □

Remark 3 (π_2^{io} vs π^{io}). We can make two remarks about the above result.

First, it would seem natural to project directly onto π^{io}, by mapping capabilities $i^{\mathbf{r}}$ and $o^{\mathbf{e}}$ into i, and $o^{\mathbf{r}}$ and $i^{\mathbf{e}}$ into o. However, the result of Proposition 2

would not hold in this case. The intuitive reason is that in doing so, we would allow two names having different sorts in $\overline{\pi}$ to be equated in the image of the encoding, thus giving rise to additional observations (since we cannot equate names having different sorts in $\overline{\pi}$). Technically, this question is reminiscent of the problem of closure of bisimilarity under substitutions in the π-calculus.

Second, the key ingredient in the definition of partial dualisation is to preserve the distinction between names having originally different sorts in the $\overline{\pi}$ process. It is possible to define an encoding of π_2^{io} into *a dyadic version* of π^{io} (without the extra capabilities), in order to do so.

Lemma 7. *Suppose $\Delta \vdash P, Q$ holds in π^{io}.*
Then $\Delta \rhd P \cong^c Q$ (in π^{io}) iff $\Delta \rhd P \cong^c Q$ (in π_2^{io}).

Proof. The right-to-left implication is immediate because any π^{io}-context is a π_2^{io}-context. To show the converse, we observe that a (Γ/Δ)-context in π_2^{io} is a (Γ'/Δ)-context in π^{io}, where Γ' is Γ where every $\underline{c}$ capability is replaced with c.

From π^{io} to $\overline{\pi}$. $\overline{\pi}$ contains π^{io}, the π-calculus with i/o-types: the rules for $\mathbf{r}$-channels are exactly those of π^{io}, and typability of $\mathbf{e}$-free processes coincides with typability in π^{io}. More precisely we can say that $\overline{\pi}$ is a conservative extension of π^{io}. In π^{io} we rely on typed barbed congruence as defined in [SW01], which is essentially the same as $\cong^c$ in $\overline{\pi}$. Before presenting the result, the following remark introduces some notation.

Remark 4. Suppose $\delta :: \Gamma \vdash P$, in π^{io}. Then $\delta^{\mathbf{r}} :: \Gamma^{\mathbf{r}} \vdash P$ in $\overline{\pi}$, where $\Gamma^{\mathbf{r}}$ stands for Γ in which all types are decorated with $\mathbf{r}$ and $\delta^{\mathbf{r}}$ stands for δ where all usages of the typing rule for restriction introduce an $\mathbf{r}$-type. Moreover $[P]^{\delta^{\mathbf{r}}} = P$.

Theorem 2 (Conservative extension). *Suppose $\Gamma \vdash P, Q$ holds in π^{io}.*
Then $\Gamma \rhd P \cong^c Q$ (in π^{io}) iff $\Gamma^{\mathbf{r}} \rhd P \cong^c Q$ (in $\overline{\pi}$).

Proof. We use π_2^{io} as an intermediate calculus. By Remark 4, let δ_P, δ_Q be derivations of $\Gamma^{\mathbf{r}} \vdash P$ and $\Gamma^{\mathbf{r}} \vdash Q$ such that $P = [P]^{\delta_P}$ and $Q = [Q]^{\delta_Q}$. By Proposition 2, the right hand side is equivalent to $(\Gamma^{\mathbf{r}})^* \rhd [P]^{\delta_P} \cong^c [Q]^{\delta_Q}$ (in π_2^{io}). By hypothesis, and since $(\Gamma^{\mathbf{r}})^* = \Gamma$, the latter is equivalent to $\Gamma \rhd P \cong^c Q$ (in π_2^{io}). Lemma 7 allows us to finish the proof. □

The result above shows that π can be embedded rather naturally into $\overline{\pi}$. This is in contrast with fusion calculi, where the equivalence on π-calculus terms induced by the embedding into fusions does not coincide with a barbed congruence or equivalence in the π-calculus.

Remark 5 ($\overline{\pi}$ and existing symmetric calculi). $\overline{\pi}$ contains the π-calculus, and hence contains (the typed version of) πI, the π-calculus with internal mobility (see [SW01]). On the other hand, because free inputs and free outputs are not allowed to interact in $\overline{\pi}$, $\overline{\pi}$ fails to represent the fusion calculus. As mentioned above, we have not succeeded in defining a 'symmetrical version' of i/o-types that would be suitable for fusions.

3 Application: Relating Encodings of the λ-calculus

In this section, we use $\overline{\pi}$ to reason about encodings of the (call-by-name) λ-calculus into the π-calculus. To do so, we need to extend $\overline{\pi}$ (Section 3.1). We then justify the validity of a transformation that makes use of link processes in Section 3.2. Finally, we show how duality, together with the latter transformation, allows us to relate Milner's encoding with the one of van Bakel and Vigliotti.

3.1 Extending $\overline{\pi}$

Based on $\overline{\pi}$, we develop an extension, called $\overline{\pi}^{\mathbf{a}}$, with forms of asynchronous communication and polyadicity. The extension to polyadic communication is standard. Asynchronous communication is added via the inclusion of *delayed* prefixes: $a(x){:}P$ (resp. $\overline{a}(x){:}P$) stands for a *(bound) delayed input (resp. output)* prefix. The intuition behind delayed prefixes is that they allow the continuation of the prefix to interact, as long as the performed action is not causally dependent on the prefix itself — this is made more precise below. Intuitively asynchrony is useful when reasoning about encodings of the λ-calculus because in a β-reduction $(\lambda x.M)N \to M[N/x]$ the "output" part N has no continuation. It is also useful to have asynchrony in input because the considered λ-strategy allows reduction under a λ-abstraction. Moreover asynchrony allows us to derive some transformation laws involving link processes (Section 3.2). Note that synchronous prefixes are still necessary, to encode the argument of an application.

Delayed prefixes are typed like bound input and output prefixes in Section 2. Types are refined with two new sorts that enforce asynchrony: $\mathbf{d}$ to force inputs to be bound and delayed, $\mathbf{a}$ to force outputs to be bound and delayed — we call such outputs *asynchronous*. For instance, if we have $a : \sharp^{\mathbf{r}}_{\mathbf{d}} T$ for some T, then all inputs at a are bound and delayed. We also include recursive types.

$$T ::= c_t\langle {}^{s_1}T_1, \ldots, {}^{s_n}T_n\rangle \mid \mathbf{1} \mid \mu X.T \mid X \qquad s ::= \mathbf{e} \mid \mathbf{r} \qquad t ::= \mathbf{d} \mid \mathbf{a}$$

In the polyadic case, $\mathbf{e}/\mathbf{r}$ sorts are given to each element of the transmitted tuple. We present here only the typing rule for delayed input, in polyadic form, to illustrate how we extend the type system of Section 2.

$$\frac{\Gamma \vdash a : i_t\langle {}^{s_1}T_1, \ldots {}^{s_n}T_n\rangle \quad \Gamma, x_1 : T_1^{s_1}, \ldots, x_n : T_n^{s_n} \vdash P}{\Gamma \vdash a(x_1, \ldots, x_n){:}P}$$

(with $T^{\mathbf{r}} = T$ and $T^{\mathbf{e}} = T^{\leftrightarrow}$). The sort $\mathbf{d}$ (resp. $\mathbf{a}$) is forbidden in the rules to type non-delayed input (resp. output) prefixes.

The definition of operational semantics is extended as follows to handle delayed prefixes (below, $\overline{\rho}(y)P$ stands for either $\overline{\rho}(y).P$ or $\overline{\rho}(y){:}P$):

$$\begin{array}{rl} P \mid \rho(x){:}Q \equiv \rho(x){:}(P \mid Q) & \text{if } x \notin \mathrm{fn}(P) \\ \rho_1(y){:}\rho_2(x){:}P \equiv \rho_2(x){:}\rho_1(y){:}P & \text{if } \mathrm{n}(\rho_1) \neq x, x \neq y, y \neq \mathrm{n}(\rho_2) \\ (\nu y)\rho(x){:}P \equiv \rho(x){:}(\nu y)P & \text{if } x \neq y, y \neq \mathrm{n}(\rho) \end{array}$$

$$\rho(x){:}(\overline{\rho}(y)P \mid Q) \to (\nu y)(P \mid Q)[y/x] \qquad \rho(x){:}P \to \rho(x){:}Q \quad \text{if } P \to Q$$
$$\rho(x){:}(\overline{\rho}b.P \mid Q) \to (P \mid Q)[b/x]$$

Barbs are defined as in Section 2, with an additional clause saying that if ρ is a barb of P and $\mathrm{n}(\rho) \neq x$, then ρ is a barb of $\rho'(x){:}P$.

The results of Section 2 hold for this extended calculus, with similar proofs:

Proposition 3 (Duality, extended calculus).

1. *Duality of typing:* $\Gamma \vdash P \Rightarrow \overline{\Gamma} \vdash \overline{P}$.
2. *Duality of barbed congruence:* $\Gamma \rhd P \cong^c Q \Rightarrow \overline{\Gamma} \rhd \overline{P} \cong^c \overline{Q}$.

The counterpart of Theorem 2 also holds in $\overline{\pi}^{\mathbf{a}}$, which stands for the extended calculus of this section, where types also specify how names have to be used in delayed prefixes. It can be stated w.r.t. $\pi^{\mathrm{io},\mathbf{a}}$, which is defined as π^{io} with additional typing information to specify which names have to be used asynchronously.

Theorem 3 (Conservative extension, extended calculus). *Suppose we have* $\Gamma \vdash P, Q$ *in* $\pi^{\mathrm{io},\mathbf{a}}$. *Then* $\Gamma \rhd P \cong^c Q$ *(in* $\pi^{\mathrm{io},\mathbf{a}}$*) iff* $\Gamma^{\mathbf{r}} \rhd P \cong^c Q$ *(in* $\overline{\pi}^{\mathbf{a}}$*)* .

The extensions $\overline{\pi}^{\mathbf{a}}$ and $\pi^{\mathrm{io},\mathbf{a}}$ are asynchronous versions of $\overline{\pi}$ and π^{io} in the sense that interaction is no longer a synchronous handshaking between two processes: for at least one of the processes, the occurrence of the interaction is not observable because the consumed action is not blocking for a continuation.

3.2 Reasoning about Links, a transformation from $o^{\mathbf{e}}_{\mathbf{a}}$ to $i^{\mathbf{r}}_{\mathbf{a}}$

The main result of this section is a technical lemma about the validity of a transformation which is used for the analysis of λ-calculus encodings in Section 3.3. A reader not interested in this result may safely skip this section.

Differently from partial dualisation (Definition 7), the transformation, written $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$, modifies prefixes, beyond simple dualisation, by introducing link processes. It also acts on types, by mapping **e**-names onto **r**-names.

Definition 8. *We set* $\langle\!\langle ab.P \rangle\!\rangle^{\mathbf{er}} = a(x).(x \twoheadrightarrow b \mid \langle\!\langle P \rangle\!\rangle^{\mathbf{er}})$, *where* $x \twoheadrightarrow b = !x(z).\overline{b}z$ *is called a* link process. *We also define* $\langle\!\langle \rho(x).P \rangle\!\rangle^{\mathbf{er}} = \rho(x).\langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$ *and similarly for delayed prefixes.* $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$ *leaves free outputs unchanged and acts homeomorphically on the other constructors.*

The transformation $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$ removes all free inputs and inserts free outputs (in the link process). We therefore expect it to return plain π processes. Moreover, the process computed in the translation of free input behaves as expected provided only the *input capability* is transmitted (the link process *at the receiver's side* exerts the input capability on x). Accordingly, we define $T_{o\mathbf{e}} = \mu X.o^{\mathbf{e}}_{\mathbf{a}}X = o^{\mathbf{e}}_{\mathbf{a}}o^{\mathbf{e}}_{\mathbf{a}}o^{\mathbf{e}}_{\mathbf{a}}\cdots$, and $T_{i\mathbf{r}} = \mu X.i^{\mathbf{r}}_{\mathbf{a}}X = i^{\mathbf{r}}_{\mathbf{a}}i^{\mathbf{r}}_{\mathbf{a}}i^{\mathbf{r}}_{\mathbf{a}}\cdots$. We let $\Gamma_{i\mathbf{r}}$ (resp. $\Gamma_{o\mathbf{e}}$) range over environments mapping all names to some $c^{\mathbf{r}}_{\mathbf{a}}T_{i\mathbf{r}}$ (resp. $c^{\mathbf{e}}_{\mathbf{a}}T_{o\mathbf{e}}$), for $c \in \{i, o, \sharp\}$.

Lemma 8 (Typing for $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$). *If* $\Gamma_{o\mathbf{e}} \vdash P$ *then* $\Gamma_{i\mathbf{r}} \vdash \langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$ *for some* $\Gamma_{i\mathbf{r}}$.

Proof. We prove by induction on P that if $\Gamma \vdash P$ then $\overline{\Gamma} \vdash \langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$. In the case for $\boldsymbol{\nu}$ we always introduce the type $\sharp^{\mathbf{r}}_{\mathbf{a}} T_{i\mathbf{r}}$. For bound prefixes we replace $c^{\mathbf{e}}_{\mathbf{a}} T_{o\mathbf{e}}$ with $\overline{c}^{\mathbf{r}}_{\mathbf{a}} T_{i\mathbf{r}}$, and for free inputs we type links with $T_{i\mathbf{r}}$ types. □

As this result shows, $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$ yields processes that only transmit the input capability. This is reminiscent of the localised π-calculus [SW01] where only the output capability is passed.

It can be noted that Lemma 8 holds because we enforce a "double contract" in the typing rules (cf. Remark 1), which allows us to typecheck bound prefixes as **e**-names (before the transformation) and as **r**-names (after).

The relationship between P and $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$ is given in terms of barbed expansion precongruence, which is a preorder in between strong and weak barbed congruence.

Definition 9 (Barbed expansion precongruence). Barbed expansion *is the largest relation* $\dot{\lesssim}$ *such that whenever* $P \dot{\lesssim} Q$,

- *if* $P \to P'$ *then* $Q \to\Rightarrow Q'$ *with* $P' \dot{\lesssim} Q'$;
- *if* $Q \to Q'$ *then* $P \to P'$ *or* $P = P'$ *with* $P' \dot{\lesssim} Q'$;
- $P \downarrow \rho$ *implies* $Q \Downarrow \rho$, *and* $Q \downarrow \rho$ *implies* $P \downarrow \rho$.

We call (resp. typed) barbed expansion precongruence ($\lesssim^c$) *the induced (resp. typed) precongruence.*

Lemma 9 (Properties of links).

1. $a : i^{\mathbf{r}}_{\mathbf{a}} T_{i\mathbf{r}}, b : o^{\mathbf{r}}_{\mathbf{a}} T_{i\mathbf{r}} \rhd a \twoheadrightarrow b \lesssim^c (\nu x)(a \twoheadrightarrow x \mid x \twoheadrightarrow b)$.
2. *If* $\Gamma_{o\mathbf{e}}, a : T_{o\mathbf{e}} \vdash P$ *then* $\Gamma_{i\mathbf{r}} \rhd \langle\!\langle P \rangle\!\rangle^{\mathbf{er}}[b/a] \lesssim^c (\nu a)(a \twoheadrightarrow b \mid \langle\!\langle P \rangle\!\rangle^{\mathbf{er}})$.

Proof. 1. The law is valid for the ordinary π-calculus (and is substitution-closed); Lemma 3 transfers the result to $\overline{\pi}$.

2. By typing, a free output involving a in $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$ is necessarily in a link; in this case, we can use (1). The other kind of interaction is with some $\overline{a}(x){:}Q$ in $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$, and $\overline{b}(x){:}Q[b/a]$ behaves like $(\boldsymbol{\nu} a)(a \twoheadrightarrow b \mid \overline{a}(x){:}Q[b/a])$. □

We use Lemma 9 to deduce operational correspondence.

Lemma 10 (Operational correspondence). *Suppose that* $\Gamma_{o\mathbf{e}} \vdash P$.

1. $P \downarrow \rho$ *iff* $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}} \downarrow \rho$.
2. *If* $P \to P'$ *then* $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}} \to\gtrsim^c \langle\!\langle P' \rangle\!\rangle^{\mathbf{er}}$.
3. *If* $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}} \to P_1$ *then* $P \to P'$ *and* $P_1 \gtrsim^c \langle\!\langle P' \rangle\!\rangle^{\mathbf{er}}$ *for some* P'.

A version of these results in the weak case can also be proved, for barbed expansion. Notably, P and $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$ exhibit the same weak barbs.

Lemma 11. *If* $\Gamma_{o\mathbf{e}} \vdash P, Q$ *then* $P \dot{\approx} Q$ *iff* $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}} \dot{\approx} \langle\!\langle Q \rangle\!\rangle^{\mathbf{er}}$.

Proof. We show that $\dot{\gtrsim}\{(\langle\!\langle P \rangle\!\rangle^{\mathbf{er}}, \langle\!\langle Q \rangle\!\rangle^{\mathbf{er}}) \mid P \dot{\approx} Q\}\dot{\lesssim}$ and $\{(P, Q) \mid \langle\!\langle P \rangle\!\rangle^{\mathbf{er}} \dot{\approx} \langle\!\langle Q \rangle\!\rangle^{\mathbf{er}}\}$ are weak barbed bisimulations. We then use the adaptation of Lemma 10 to the weak case, for barbed expansion. □

Lemma 12. *If* $\Gamma_{\mathbf{oe}} \vdash P, Q$ *and* $\Gamma_{\mathbf{ir}} \rhd \langle\!\langle P \rangle\!\rangle^{\mathbf{er}} \cong^c \langle\!\langle Q \rangle\!\rangle^{\mathbf{er}}$ *then* $\Gamma_{\mathbf{oe}} \rhd P \cong^c Q$.

Proof. We define a type system with marks on types, such that only $T_{\mathbf{ir}}$-types are marked. The marking propagates onto the names of the typed processes. We modify the encoding $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$ to only operate on marked prefixes. For every $(\Delta/\Gamma_{\mathbf{oe}})$-context C, its encoding $\langle\!\langle C \rangle\!\rangle^{\mathbf{er}}$ is a $(\Delta'/\Gamma_{\mathbf{ir}})$-context. Thanks to the compositionality of $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$, the hypothesis of the lemma implies the equivalence $\langle\!\langle C[P] \rangle\!\rangle^{\mathbf{er}} \mathrel{\dot{\approx}} \langle\!\langle C[Q] \rangle\!\rangle^{\mathbf{er}}$. We then adapt the proof of Lemma 11 to this marked encoding. □

3.3 An analysis of van Bakel and Vigliotti's encoding

As announced in Section 1, we start from an adaptation of Milner's call-by-name (cbn) encoding of [Mil92] to *strong* cbn, which also allows reductions to occur under λ. We obtain this by using a delayed prefix in the clause for λ-abstraction. The encoding, noted $[\![\cdot]\!]^{\mathcal{M}}$, is defined as follows:

$$[\![x]\!]^{\mathcal{M}}_p = \overline{x}p \qquad [\![\lambda x.M]\!]^{\mathcal{M}}_p = p(x,q).[\![M]\!]^{\mathcal{M}}_q$$
$$[\![MN]\!]^{\mathcal{M}}_p = (\nu q)\big([\![M]\!]^{\mathcal{M}}_q \mid (\nu x)(\overline{q}\langle x, p\rangle \mid !x(r).[\![N]\!]^{\mathcal{M}}_r)\big)$$

The other encoding we analyse, taken from [vBV09], is written $[\![\cdot]\!]^{\mathcal{B}}$:

$$[\![x]\!]^{\mathcal{B}}_p = x(p').p' \rightarrowtriangle p \qquad [\![\lambda x.M]\!]^{\mathcal{B}}_p = \overline{p}(x,q).[\![M]\!]^{\mathcal{B}}_q$$
$$[\![MN]\!]^{\mathcal{B}}_p = (\nu q)\big([\![M]\!]^{\mathcal{B}}_q \mid q(x,p').(p' \rightarrowtriangle p \mid !\overline{x}(r).[\![N]\!]^{\mathcal{B}}_r)\big)$$

Note that $[\![\cdot]\!]^{\mathcal{B}}$ is written in [vBV09] using asynchronous free output and restriction instead of delayed bound output. We can adopt this more concise notation since $(\nu x)(\overline{a}x \mid P)$ and $\overline{a}(x).P$ are strongly bisimilar processes, and similarly for $x(p').p' \rightarrowtriangle p$ and $x(p').p' \rightarrowtriangle p$. (Another difference is that the replication in the encoding of the application is guarded, as in [vBV10], to force a tighter operational correspondence between reductions in λ and in the encodings.)

As remarked above, $[\![\cdot]\!]^{\mathcal{B}}$ and $[\![\cdot]\!]^{\mathcal{M}}$ differ considerably because they engage in quite different dialogues with their environments: in $[\![\cdot]\!]^{\mathcal{M}}$ a function receives its argument via an input, in $[\![\cdot]\!]^{\mathcal{B}}$ it interacts via an output. Differences are also visible in the encodings of variables and application (e.g. the use of links).

To compare the encodings $[\![\cdot]\!]^{\mathcal{M}}$ and $[\![\cdot]\!]^{\mathcal{B}}$, we introduce an intermediate encoding, noted $[\![\cdot]\!]^{\mathcal{I}}$, which is defined as the dual of $[\![\cdot]\!]^{\mathcal{M}}$ (in $\overline{\pi}$):

$$[\![x]\!]^{\mathcal{I}}_p = xp \qquad [\![\lambda x.M]\!]^{\mathcal{I}}_p = \overline{p}(x,q).[\![M]\!]^{\mathcal{I}}_q$$
$$[\![MN]\!]^{\mathcal{I}}_p = (\nu q)\big([\![M]\!]^{\mathcal{I}}_q \mid (\nu x)(q\langle x, p\rangle \mid !\overline{x}(r).[\![N]\!]^{\mathcal{I}}_r)\big)$$

Note that while $[\![\cdot]\!]^{\mathcal{M}}$ and $[\![\cdot]\!]^{\mathcal{B}}$ can be expressed in π, $[\![\cdot]\!]^{\mathcal{I}}$ uses free input, and does thus not define π-calculus processes.

The three encodings given above are based on a similar usage of names. Two kinds of names are used: we refer to names that represent continuations (p, p', q, r in the encodings) as *handles*, and to names that stand for λ-calculus parameters (x, y, z) as *λ-variables*. Here is how these encodings can be typed in $\overline{\pi}$:

Lemma 13 (Typing the encodings). $[\![\cdot]\!]^{\mathcal{M}}$, $[\![\cdot]\!]^{\mathcal{B}}$ *and* $[\![\cdot]\!]^{\mathcal{I}}$ *yield processes which are typable with the respective typing environments* $\Gamma_{\mathcal{M}}, \Gamma_{\mathcal{B}}, \Gamma_{\mathcal{I}}$*, where:*

- $\Gamma_{\mathcal{M}}$ *types* λ*-variables with* $o^{\mathbf{r}}_{\mathbf{a}}H$ *and handles with* $H = \mu X.i^{\mathbf{r}}_{\mathbf{d}}\langle o^{\mathbf{r}}_{\mathbf{a}}X, X\rangle$*;*
- $\Gamma_{\mathcal{B}}$ *uses respectively* $i^{\mathbf{r}}_{\mathbf{d}}G$ *and* $o^{\mathbf{r}}_{\mathbf{a}}\langle o^{\mathbf{r}}_{\mathbf{d}}G, G\rangle$ *where* $G = \mu Y.i^{\mathbf{r}}_{\mathbf{a}}\langle o^{\mathbf{r}}_{\mathbf{d}}Y, Y\rangle$*;*
- $\Gamma_{\mathcal{I}}$ *is the dual of* $\Gamma_{\mathcal{M}}$ *(that is, it uses* $i^{\mathbf{e}}_{\mathbf{d}}\overline{H}$ *and* $\overline{H} = \mu Z.o^{\mathbf{e}}_{\mathbf{a}}\langle i^{\mathbf{e}}_{\mathbf{d}}Z, Z\rangle$*).*

Encoding $[\![\cdot]\!]^{\mathcal{I}}$ can be obtained from $[\![\cdot]\!]^{\mathcal{M}}$ by duality. The only difference between $[\![\cdot]\!]^{\mathcal{I}}$ and $[\![\cdot]\!]^{\mathcal{B}}$ is the presence of two links. We rely on a link transformation similar to the one of Section 3.2 to move from $[\![\cdot]\!]^{\mathcal{I}}$ to $[\![\cdot]\!]^{\mathcal{B}}$. Thus, by composing the results on duality and on the transformation, we are able to go from $[\![\cdot]\!]^{\mathcal{M}}$ to $[\![\cdot]\!]^{\mathcal{B}}$.

Proposition 4. *Given two* λ*-terms* M *and* N*, we have* $[\![M]\!]^{\mathcal{M}}_p \dot{\approx} [\![N]\!]^{\mathcal{M}}_p$ *if and only if* $[\![M]\!]^{\mathcal{B}}_p \dot{\approx} [\![N]\!]^{\mathcal{B}}_p$ *(both equivalences are in* $\pi^{\mathrm{io},\mathbf{a}}$*).*

Proof. By duality, $[\![M]\!]^{\mathcal{M}}_p \dot{\approx} [\![N]\!]^{\mathcal{M}}_p$ iff $[\![M]\!]^{\mathcal{I}}_p \dot{\approx} [\![N]\!]^{\mathcal{I}}_p$. To establish that this is equivalent to $[\![M]\!]^{\mathcal{B}}_p \dot{\approx} [\![N]\!]^{\mathcal{B}}_p$, we rely on an adaptation of Lemma 11. For this, we define a transformation that exploits the ideas presented in Section 3.2. In particular, handles (p, p', q, r) are treated like in Definition 8. The handling of λ-variables (x, y, z) is somehow orthogonal, and raises no major difficulty, because such names are always transmitted as bound (fresh) names. □

Remark 6 (Call by name). To forbid reductions under λ-abstractions, we could adopt Milner's original encoding, and use an input prefix instead of delayed input in the translation of abstractions. Accordingly, adapting Van Bakel and Vigliotti's encoding to this strategy would mean introducing a free input prefix — which is rather natural in $\overline{\pi}$, but is not in the π-calculus.

4 Concluding remarks

We have presented several properties of $\overline{\pi}$, and established relationships with the π-calculus with i/o-types (π^{io}).

The calculus $\overline{\pi}$ enjoys properties of dualities while being "large", in the sense that it incorporates many of the forms of prefix found in dialects of the π-calculus (free input, bound input, and, in the extension in Section 3.1, also delayed input, plus the analogue for outputs), and a non-trivial type system based on i/o-types. This syntactic abundance makes $\overline{\pi}$ a possibly interesting model in which to study various forms of dualities. This is exemplified in our study of encodings of the λ-calculus, where we have applied $\overline{\pi}$ and its theory to explain a recent encoding of cbn λ-calculus by van Bakel and Vigliotti: it can be related, via dualities, to Milner's encoding.

It would be interesting to strengthen the full abstraction in Lemma 11 from barbed bisimilarity to barbed congruence. This would allow us to replace barbed bisimilarity with typed barbed congruence in Proposition 4 as well (using the type environments of Lemma 13). While we believe the result to be true, the

proof appears difficult because the link transformation modifies both processes and types, so that the types needed for barbed congruence in the two encodings are different. Therefore also the sets of contexts to be taken into account are different. The problem could be tackled by combining the theory on delayed input and the link bisimilarity in [MS04], and adapting it to a typed setting.

We plan to further investigate the behavioural theory of $\overline{\pi}$, and study in particular other transformations along the lines of Section 3.2, where link processes are used to implement substitutions. It would be interesting to provide general results on process transformations in terms of links, when the direction and the form of the links vary depending on the types of the names involved. Currently we only know how to handle them when the calculus is asynchronous and localised [MS04].

As already mentioned, another interesting issue is how to accommodate i/o-types into πI and fusion calculi while maintaining the dualities of the untyped calculi.

Acknowledgments. This work was supported by the french ANR projects Recre, 2009-BLAN-0169-02 Panda, and 2010-BLAN-0305-01 PiCoq.

References

[Fu97] Y. Fu. The χ-calculus. In *Proc. of APDC'97*, pages 74–81. IEEE Computer Society Press, 1997.

[GW00] P. Gardner and L. Wischik. Explicit fusions. In *Proc. of MFCS*, volume 1893 of *LNCS*, pages 373–382. Springer-Verlag, 2000.

[HMS12] D. Hirschkoff, J.M. Madiot, and D. Sangiorgi. On subtyping in symmetric versions of the π-calculus. In preparation, 2012.

[Mer00] M. Merro. *Locality in the pi-calculus and applications to distributed objects.* PhD thesis, École des Mines, France, 2000.

[Mil92] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[MS04] M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. *Mathematical Structures in Computer Science*, 14(5):715–767, 2004.

[PV98] J. Parrow and B. Victor. The fusion calculus: expressiveness and symmetry in mobile processes. In *Proc. of LICS*, pages 176 –185. IEEE, 1998.

[San96] D. Sangiorgi. π-calculus, internal mobility, and agent-passing calculi. In *Selected papers from TAPSOFT '95*, pages 235–274. Elsevier, 1996.

[SW01] D. Sangiorgi and D. Walker. *The Pi-Calculus: a theory of mobile processes.* Cambridge University Press, 2001.

[Vas09] V. T. Vasconcelos. Fundamentals of session types. In *Proc. of SFM*, volume 5569 of *LNCS*, pages 158–186. Springer, 2009.

[vBV09] S. van Bakel and M. G. Vigliotti. A logical interpretation of the λ-calculus into the π-calculus, preserving spine reduction and types. In *Proc. of CONCUR*, volume 5710 of *LNCS*, pages 84–98. Springer, 2009.

[vBV10] S. van Bakel and M. G. Vigliotti. Implicative logic based encoding of the λ-calculus into the π-calculus, 2010. From http://www.doc.ic.ac.uk/ svb/.