A behavioural theory for a π-calculus with preorders

Daniel Hirschkoff[1], Jean-Marie Madiot[1], and Xian Xu[2]

[1] ENS Lyon, Université de Lyon, CNRS, INRIA, France,
[2] East China University of Science and Technology, China

Abstract. We study the behavioural theory of $\pi\mathtt{P}$, a π-calculus in the tradition of Fusions and Chi calculi. In contrast with such calculi, reduction in $\pi\mathtt{P}$ generates a preorder on names rather than an equivalence relation. We present two characterisations of barbed congruence in $\pi\mathtt{P}$: the first is based on a compositional LTS, and the second is an axiomatisation. The results in this paper bring out basic properties of $\pi\mathtt{P}$, mostly related to the interplay between the restriction operator and the preorder on names.
Consequently, $\pi\mathtt{P}$ is a calculus in the tradition of Fusion calculi, in which both types and behavioural equivalences can be exploited in order to reason rigorously about concurrent and mobile systems.

1 Introduction

The π-calculus expresses mobility via name passing, and has two binders: the input prefix binds the value to be received, and restriction is used to delimit the scope of a private name. The study of Fusions [16], Chi [8], Explicit Fusions [20] and Solos [13] has shown that using restriction as the only binder is enough to express name passing. In such calculi (which, reusing a terminology from [10], we shall refer to as *fusion calculi*), the bound input prefix, $c(x).P$, is dropped in favour of free input, $cb.P$, and communication involving two prefixes cb and $\overline{c}a$ generates the *fusion* of names a and b. This yields a pleasing symmetry between input and output prefixes; moreover, one can encode bound input in terms of free input as $(\nu x)cx.P$. Fusion calculi therefore promote *minimality* (keep only restriction as a binder) and *symmetry* (input and output prefixes play similar roles). Moreover, and most importantly, fusions act on restricted names, in contrast with the π-calculus, where restricted names can only replace names bound by input (and are thus treated like constants).

The behavioural theory of existing fusion calculi is generally simpler than in the π-calculus (in particular, bisimilarity is a congruence). Fusion calculi have notably been used to analyse concurrent constraints [19], to study distributed implementations of programming languages [9,5] and to establish connections with proof theory [7].

Symmetry comes however at a price. It has indeed been shown in [10] that i/o-types (input/output types, [17]) cannot be adapted to a fusion calculus. Such

types go beyond the simple discipline of sorting, and can be useful, in particular, to reason using *typed behavioural equivalences* [17,18].

The intuitive reason of the incompatibility of i/o-types with fusions can be explained by considering the following structural congruence law in Explicit Fusions (but the point is essentially the same for other fusion calculi):

$$a(x).P \mid a{=}b \;\equiv\; b(x).P \mid a{=}b \;\;.$$

Process $a{=}b$ is an explicit fusion. The law says that in presence of $a{=}b$, an input on a can be viewed as the same input on b, *and vice-versa* (fusion processes are somehow akin to equators, in an asynchronous setting [12]). This shows that fusions define a symmetric relation on names; this is incompatible with a nontrivial (i.e., asymmetric) subtyping relation, which is necessary for i/o-types to make sense.

This observation has led in [10] to the introduction of $\pi\mathtt{P}$, a π-calculus with name preorders. The most important difference between $\pi\mathtt{P}$ and existing name-passing calculi is that interaction does not have the effect of equating (or *fusing*) two names, but instead generates an *arc* process, as follows:

$$\overline{c}a.P \mid cb.Q \quad\longrightarrow\quad a\!\not\!b \mid P \mid Q \;\;.$$

The arc $a\!\not\!b$ expresses the fact that anything that can be done using name b can be done using a as well (but not the opposite): we say that a is *above* b. Arcs induce a preorder relation on names, which can evolve along reductions.

Arcs can modify interaction possibilities: in presence of $a\!\not\!b$, a is above b, hence a process emitting on b can also make an output transition along channel a. In general, an output on channel c can interact with an input on d provided c and d are *joinable*, written $c \curlyvee d$, which means that there is some name that is above both c and d according to the preorder relation. To formalise these observations, the operational semantics exploits *conditions* involving names, which are either of the form $b \prec a$ (a is above b), or $a \curlyvee b$ (a and b are joinable).

$\pi\mathtt{P}$ can be described as a variant of Explicit Fusions, in which arcs replace fusion processes. Beyond the possibility to define i/o-types and subtyping for $\pi\mathtt{P}$ [10], we would like to analyse the consequences of the novel aspects of $\pi\mathtt{P}$, whose behaviour does not seem to be reducible to existing calculi.

In particular, name preorders have an impact on how processes express behaviours. Barbed congruence for $\pi\mathtt{P}$, written $\simeq$, is defined in [10]. Some laws for $\simeq$ suggest that the behavioural theory of $\pi\mathtt{P}$ differs w.r.t. existing fusion calculi. As an illustration, consider the following *interleaving law*, which is valid in $\pi\mathtt{P}$ (and in π):

$$\overline{a}(x).\overline{b}(y).(\overline{x} \mid y) \;\simeq\; \overline{a}(x).\overline{b}(y).(\overline{x}.y + y.\overline{x}) \;\;.$$

$\overline{a}(x)$ is the emission of a fresh name x on a, and $\overline{x}$ (resp. y) stands for an output (resp. input) where the value being transmitted is irrelevant. In Fusions, unlike in the π-calculus, the process that creates successively two fresh names x and y cannot prevent the context from equating ("fusing") x and y. Hence, in order for the equivalence to hold, it is necessary to add a third summand on the right, $[x = y]\tau$. This example suggests that $\pi\mathtt{P}$ gives a better control on restricted

names than existing fusion calculi. This issue also motivated the study of two variants of fusion calculi that have a refined notion of restriction [3,4].

The main purpose of the present work is to deepen the study of the behavioural theory of $\pi\mathtt{P}$, in an untyped setting. We define a Labelled Transition System (LTS) for $\pi\mathtt{P}$, and show that the induced notion of bisimilarity, written $\sim$, characterises $\simeq$ (Section 3). It can be noted that [10] presents a characterisation of barbed congruence, using an LTS that is rather ad hoc, because it is based on the definition of the reduction relation. Unlike the latter, the LTS we present here is *structural*.

The LTS reveals interesting aspects of interaction in $\pi\mathtt{P}$. An important observation is related to the interplay between arcs and the restriction operator. It is for instance possible for a process to react to an input offer on some channel, say c, without being actually able to perform an output on c. This is the case for process $P_0 \triangleq (\nu a)(\overline{a}(x).0 \mid a/\!\!c)$. Because a is above c in the preorder, P_0 cannot do an output on c, although c is occurs free in P_0 (it could if the arc $a/\!\!c$ was replaced with $c/\!\!a$). However, $P_0 \mid c(y).0$ can perform a reduction: intuitively, by extending the scope of (νa), the input at c can be moved to a, so that the communication takes place.

This phenomenon leads to the addition of a new type of labels in the LTS, corresponding to what we call *protected actions*: in the example P_0 can do a protected output at c, meaning that it can react to an input offer at c. Accordingly, we introduce *protected names*, which correspond to (usages of) names where a protected action occurs: intuitively, in P_0, name c is protected. As expected, protected actions correspond to observables in the reduction-based semantics supporting the definition of $\simeq$.

Arc processes do not have transitions, but they induce relations between names, which in turn influence the behaviour of processes. Accordingly, strong bisimilarity, $\sim$, not only tests transitions, but also has a clause to guarantee that related processes entail the same conditions.

Finally, the LTS also includes a label $[\varphi]\tau$, expressing "conditional synchronisation". Intuitively, process $\overline{a} \mid b$ is not able to perform a τ transition by itself, but it should be when the environment entails $a \curlyvee b$. Hence, in order for our LTS to be compositional, we include labels of the form $[\varphi]\tau$, interpreted as "τ under the condition φ".

In Section 4, we provide a second characterisation of barbed congruence, by presenting a set of laws that define an axiomatisation of $\simeq$. Algebraic laws help analysing the behaviour of the constructs of the calculus and their interplay. We present a sample of behavioural equalities, and explain how they can be derived equationally, in Section 4.1.

The axiomatisation we give is less simple than, say, the one for Fusions in [16], for two reasons: first, we manipulate preorders between names rather than equivalences. Second, the preorder is explicitly represented in processes, so that some equational laws must describe the interplay between processes and the preorder relation. On the contrary, such aspects are dealt with implicitly in Fusions—we sketch how our ideas can be adapted to Explicit Fusions in Section 4.3.

The axiomatisation exploits the idea that $\pi\mathtt{P}$ processes have a *state* component, corresponding to the preorder induced by arcs. Several laws in the axiomatisation express persistence of the state component (the state can only be extended along computation). Moreover, the restriction operator prevents the state from being globally shared in general: for instance, in process P_0 above, name a can be used instead of c, but is only known inside the scope of (νa). All in all, the handling of restriction in our axiomatisation requires more care than is usually the case, due to the necessity to express the "view" that subprocesses have on the preorder of names.

To present the axiomatisation, we renounce minimality. The syntax of the calculus in this paper differs from the one in [10]: we include bound prefixes and sums with conditions, as it is customary for axiomatisations for the π-calculus [15,18]. We compare the calculus from [10] with ours in Remark 3 and Proposition 11. We show that the differences are unimportant: the calculus from [10] can be encoded into ours and the behavioural equivalence is unaffected.

We focus in this paper on a finite calculus. This is sufficient to enlighten the main aspects of the behavioural theory of processes. We do not expect any unpredicted difficulty to arise, in the definition of labelled transitions and bisimilarity, from the extension of $\pi\mathtt{P}$ with a replication operator.

The paper describes our results and sketches the most important proofs. We refer to [11] for a more detailed presentation of the technical details. Related work is discussed along the paper, where it is relevant.

2 $\pi\mathtt{P}$: Reduction-Based Semantics

The Calculus: Preorders and Processes. We consider a countable set of names $a, b, c, \ldots, x, y, \ldots$, and define conditions (φ), extended names (α, β), prefixes (π) and processes (P, Q) as follows:

$$\varphi ::= a \prec b \mid a \curlyvee b \qquad \alpha, \beta ::= a \mid \{a\} \qquad \pi ::= \alpha(x) \mid \overline{\alpha}(x) \mid [\varphi]\tau$$
$$P, Q ::= P \mid Q \mid (\nu a)P \mid a\not b \mid \Sigma_{i\in I}\pi_i.P_i$$

There are two forms of conditions, ranged over with φ: $\varphi = a \prec b$ is read "b is above a" and $\varphi = a \curlyvee b$ is read "a and b are joinable". In both cases, we have $\mathsf{n}(\varphi) = \{a, b\}$. We explain below how we extend relations $\prec$ and $\curlyvee$ to extended names. When $\mathsf{n}(\varphi) = \{a\}$, we say that φ is *reflexive*, and abbreviate in this case prefix $[\varphi]\tau$ as τ. Condition $b \prec a$ is ensured by the arc process $a\not b$.

In a prefix $\alpha(x)$ or $\overline{\alpha}(x)$, we say that extended name α is in subject position, while x is in object position. As discussed in Section 1, extended names include *protected names*, of the form $\{a\}$, which can be used in subject position only. We call *protected prefix* a prefix where the subject is a protected name. A prefix of the form $[\varphi]\tau$ is called a *conditional* τ, while other prefixes are called *visible*. Bound and free names for prefixes are given by: $\mathsf{bn}([\varphi]\tau) = \emptyset$ and $\mathsf{bn}(\alpha(x)) = \mathsf{bn}(\overline{\alpha}(x)) = \{x\}$, $\mathsf{fn}([\varphi]\tau) = \mathsf{n}(\varphi)$, $\mathsf{fn}(\alpha(x)) = \mathsf{fn}(\overline{\alpha}(x)) = \mathsf{n}(\alpha)$ with $\mathsf{n}(a) = \mathsf{n}(\{a\}) = \{a\}$.

In a sum process, we let I range over a finite set of integers. 0 is the inactive process, defined as the empty sum. We use S to range over sum processes of the

form $\Sigma_{i\in I}\pi_i.P_i$, and write $\pi.P \in S$ if $\pi.P$ is a summand of S. We sometimes decompose sum processes using the binary sum operator, writing, e.g., $S_1 + S_2$ (in particular, $S + 0 = S$). We abbreviate $\pi.0$ as π, and write $\alpha(x).P$ simply as $\alpha.P$ when the transmitted name is not relevant, and similarly for $\overline{\alpha}$. In $(\nu a)P$, (νa) binds a in P, and prefixes $\alpha(x)$ and $\overline{\alpha}(x)$ bind x in the continuation process. The set of free names of P, $\mathsf{fn}(P)$, is defined in the usual way, and we work up to α-conversion of processes. $P\{^b/_a\}$ is the process obtained by substituting a with b in P, in a capture-avoiding way.

We use an overloaded notation, and define processes representing conditions:

$$a \curlyvee b \triangleq (\nu u)(u/a \mid u/b) \qquad\qquad a \prec b \triangleq b/a \ .$$

Below, Γ ranges over sets of conditions. We define $\Gamma \vdash \varphi$, meaning that Γ implies φ, and $P \rhd \varphi$ (we write $P \rhd \Gamma$ to express that P entails φ for all $\varphi \in \Gamma$):

$$\frac{}{\Gamma \vdash a \prec a} \quad \frac{\varphi \in \Gamma}{\Gamma \vdash \varphi} \quad \frac{\Gamma \vdash b \curlyvee a}{\Gamma \vdash a \curlyvee b} \quad \frac{\Gamma \vdash a \prec b \quad \Gamma \vdash b \prec c}{\Gamma \vdash a \prec c} \quad \frac{\Gamma \vdash a \prec b \quad \Gamma \vdash c \prec b}{\Gamma \vdash a \curlyvee c} \quad \frac{\Gamma \vdash a \prec b \quad \Gamma \vdash b \curlyvee c}{\Gamma \vdash a \curlyvee c}$$

$$\frac{}{a/b \rhd b \prec a} \quad \frac{P \rhd \Gamma \quad \Gamma \vdash \varphi}{P \rhd \varphi} \quad \frac{P \rhd \varphi}{P \mid Q \rhd \varphi} \quad \frac{Q \rhd \varphi}{P \mid Q \rhd \varphi} \quad \frac{P \rhd \varphi \quad a \notin \mathsf{n}(\varphi)}{(\nu a)P \rhd \varphi}$$

As an example, the reader might check that $(\nu u)(u/a \mid u/b) \mid b/c \rhd a \curlyvee c$.

Reduction Semantics and Barbed Congruence. The definition of structural congruence, $\equiv$, is standard. In particular, we have

$$\Sigma_{i\in I}\pi_i.P_i \ \equiv \ \Sigma_{i\in I}\pi_{\sigma(i)}.P_{\sigma(i)} \quad \text{if } \sigma \text{ is a permutation of } I \ .$$

Relations $\equiv$ and $\rhd$ are used to define the reduction of processes. We rely on $\rhd$ to infer that two processes interact on joinable (extended) names. This allows us to introduce reduction-closed barbed congruence, along the lines of [10].

Definition 1 (Reduction). *Relation $\mapsto$ is defined by the following rules:*

$$\frac{\overline{\alpha}(x).P \in S_1 \quad \beta(y).Q \in S_2 \quad R \rhd \alpha \curlyvee \beta \quad x \neq y}{R \mid S_1 \mid S_2 \ \mapsto \ R \mid (\nu xy)(x/y \mid P \mid Q)}$$

where:
$a \curlyvee \{b\} = \{b\} \curlyvee a = a \prec b$
$\{a\} \curlyvee \{b\} =$ undefined

$$\frac{[\varphi]\tau.P \in S \qquad R \rhd \varphi}{R \mid S \ \mapsto \ R \mid P} \qquad \frac{P \mapsto P'}{P \mid R \mapsto P' \mid R} \qquad \frac{P \mapsto P'}{(\nu a)P \mapsto (\nu a)P'} \qquad \frac{P \equiv \mapsto \equiv P'}{P \mapsto P'}$$

Definition 2 (Barbs, barbed congruence). *We write $P \downarrow_{\overline{a}}$ if $P \mid a(x).\omega \mapsto P'$, where P' is a process in which ω is unguarded, and ω is a special name that does not appear in P. We define similarly the barb $\downarrow_a$, using the tester $\overline{a}(x).\omega$.*

Barbed congruence, $\simeq$, is the largest congruence that satisfies:

- *if $P \downarrow_a$ and $P \simeq Q$ then $Q \downarrow_a$, and similarly for $\downarrow_{\overline{a}}$, and*
- *if $P \mapsto P'$ and $P \simeq Q$ then for some Q', $Q \mapsto Q'$ and $P' \simeq Q'$.*

We can remark that $P_0 \downarrow_{\overline{c}}$, where P_0 is the process defined in Section 1.

The remainder of the paper is devoted to the presentation of two characterisations of $\simeq$. We first comment on the definition of $\pi\mathtt{P}$ given above.

One could consider an alternative version of reduction, called "eager", whereby arcs can rewrite prefixes in one step of computation, yielding, e.g., $d\!/\!c \mid c(x).P \mapsto d\!/\!c \mid d(x).P$. It appears in [10] that the present semantics is more compelling (for instance $a(x).a(y)$ would not be equivalent to $a(x) \mid a(y)$ in the eager version).

Remark 3 (Encodability of free and protected prefixes).

In $\pi\mathtt{P}$, arcs act like "instantaneous forwarders". This allows us to define an encoding $[\cdot]_{\mathrm{f}}$ from a calculus with free prefixes to a calculus with bound prefixes as follows (x is chosen fresh):

$$[ab.P]_{\mathrm{f}} \triangleq a(x).([P]_{\mathrm{f}} \mid x\!/\!b) \qquad [\overline{a}b.P]_{\mathrm{f}} \triangleq \overline{a}(x).([P]_{\mathrm{f}} \mid b\!/\!x) \ ,$$

where $[\cdot]_{\mathrm{f}}$ preserves other operators of the calculi. We return to this encoding below (Proposition 11), and show that it allows us to reflect behavioural equivalence in [10] into our calculus.

We can also encode protected prefixes as follows (u is chosen fresh):

$$[\{a\}(x).P]_{\mathrm{p}} \triangleq (\nu u)(u\!/\!a \mid u(x).[P]_{\mathrm{p}}) \qquad [\{\overline{a}\}(x).P]_{\mathrm{p}} \triangleq (\nu u)(u\!/\!a \mid \overline{u}(x).[P]_{\mathrm{p}}) \ .$$

Although protected prefixes are in some sense redundant, we do not treat them as derived operators, to simplify the presentation (in particular in Section 4).

The results of this paper (Sections 3 and 4) can be adapted to a calculus featuring only free prefixes, and restriction as the only binder, like the calculus of [10]. This yields more complex definitions to handle bound prefixes and protected actions, in particular when defining sum processes. We discuss in [11] a presentation of transitions and bisimilarity based on free prefixes. It can be noted that the axiomatisation of Fusions given in [16] relies only on free input and output, and treats bound prefixes as derived operators. We think that, for $\pi\mathtt{P}$, handling prefixes for bound and protected actions as derived operators would introduce further technical complications that would make the axiomatisation more obscure.

3 A Labelled Transition System for $\pi\mathtt{P}$

3.1 LTS and Bisimilarity

The LTS defines transitions $P \xrightarrow{\mu} P'$, where the grammar for the labels, μ, is the same as the one for prefixes π. We comment on the rules, given in Figure 1.

The first two rules correspond to the firing of visible prefixes. The transition involves a fresh name x, upon which the participants in a communication "agree". Name y remains local, via the installation of an arc, according to the directionality of the prefix. (Adopting a rule with no arc installation would yield a more complex definition of $\sim$). The rule for the $[\varphi]\tau$ prefix is self explanatory. The rule describing communication follows the lines of the corresponding rule for $\mapsto$; no arc is installed (but arcs are introduced in the prefix rules).

The three rules mentioning $\rhd$ are called *preorder rules*. The two preorder rules for visible actions exploit $\prec$, which is defined for extended names (as we

did for $\curlyvee$ above). Note that the condition involving $\rhd$ is *the same* in these two rules. To understand these rules, and the role of protected actions, we recall the basic intuition about arcs: an arc $d/\!\!/a$ can transform an interaction at a into an interaction at d. For instance, from $P \xrightarrow{a(x)} P'$ and $P \rhd a \prec d$, we can derive $P \xrightarrow{d(x)} P'$. As a consequence, an input at a can synchronise with an output at b if both a and b can be "pulled upwards in the preorder", using arcs, to some name, say u, which is above a and b. Observe also that if, like in P above, the input at a is transformed into an input at d, then a name u' standing above d and b can be used to let the synchronisation happen (because u' would be above a and b).

If, on the contrary, we want to replace, in the input, name a with a name that sits below a, say c (like in process P_0 from Section 1), we are moving *downwards* in the preorder. Because of this, the action becomes protected, and we can derive for instance $P_0 \xrightarrow{\overline{\{c\}}(y)}$, because $\overline{a}(x).0 \mid a/\!\!/c \rhd c \curlyvee a$ (and hence $a \prec \{c\}$). By going downwards, we have somehow fixed the channel where the communication occurs (e.g., at a in $Pf_x u0$). Indeed, it is no longer the case that an output at b can synchronise with the protected input at c whenever some u is above b and c, because such u would not necessarily be above a (where the original input takes place) and b in the preorder. For this reason, we can only move further downwards in the preorder, and for instance deduce, from $P_0 \xrightarrow{\overline{\{c\}}(y)}$, that $P_0 \xrightarrow{\overline{\{c_1\}}(y)}$ as soon as $c_1 \prec c$ (which implies $\{c\} \prec \{c_1\}$).

The other preorder rule can be used to modify conditional τs involved in a transition. As an example, let $P_1 \triangleq (\overline{a}(x).Q \mid n/\!\!/u) \mid (u(y).R \mid n/\!\!/a)$. Process P_1 can perform a τ transition: the two arcs can, intuitively, let the output at a and the input at u interact at name n. Technically, this can be derived by inferring a $\xrightarrow{[a \curlyvee u]\tau}$ transition (from the output on the left and the input on the right), which can then be turned into a τ transition, exploiting the fact that *the whole process* entails $a \curlyvee u$. Finally, the congruence rules are as expected.

Definition 4 ($\sim$). *A symmetric relation $\mathcal{R}$ is a bisimulation if $P \mathcal{R} Q$ implies:*

- *If $P \rhd \varphi$ then $Q \rhd \varphi$.*
- *If $P \xrightarrow{\alpha(x)} P'$, with $x \notin \mathsf{fn}(Q)$, then there is Q' such that $Q \xrightarrow{\alpha(x)} Q'$ and $P' \mathcal{R} Q'$; we impose the same condition with $\overline{\alpha}$ instead of α.*
- *If $P \xrightarrow{[\varphi]\tau} P'$ then there is Q' such that $Q \xrightarrow{[\varphi]\tau} Q'$ and $P' \mid \varphi \mathcal{R} Q' \mid \varphi$.*

Bisimilarity, written $\sim$, is the greatest bisimulation.

This definition can be related to the efficient bisimulation from [20]. In the last clause, we add φ in parallel, since the transition is fired only if φ is satisfied.

Remark 5. Our LTS does not have rules for opening and closing the scope of a restriction. Instead, we rely on arcs in $\pi\mathtt{P}$ to handle scope extrusion. To illustrate this, consider the following $\pi\mathtt{P}$ transition where a a private name c is emitted:

$$\overline{a}(c).P \xrightarrow{\overline{a}(x)} (\nu c)(c/\!\!/x \mid P) \ .$$

Name x is visible in the label, and arc $c/\!\!/x$ is installed. Through x, the environment can affect c, so that $\pi\mathtt{P}$ actually *implements* scope extrusion via arcs,

$$\frac{x \notin \mathsf{n}(\alpha) \cup \{y\} \cup \mathsf{fn}(P)}{\alpha(y).P \xrightarrow{\alpha(x)} (\nu y)(x/y \mid P)} \qquad \frac{x \notin \mathsf{n}(\alpha) \cup \{y\} \cup \mathsf{fn}(P)}{\overline{\alpha}(y).P \xrightarrow{\overline{\alpha}(x)} (\nu y)(y/x \mid P)} \qquad \frac{}{[\varphi]\tau.P \xrightarrow{[\varphi]\tau} P}$$

$$\frac{P \xrightarrow{\overline{\alpha}(x)} P' \qquad Q \xrightarrow{\beta(x)} Q'}{P \mid Q \xrightarrow{[\alpha \curlyvee \beta]\tau} (\nu x)(P' \mid Q')} \qquad \frac{P \xrightarrow{[\varphi_2]\tau} P' \quad P \rhd \Gamma \quad \Gamma, \varphi_1 \vdash \varphi_2}{P \xrightarrow{[\varphi_1]\tau} P'}$$

$$\frac{P \xrightarrow{\alpha(x)} P' \quad P \rhd \alpha \prec \beta}{P \xrightarrow{\beta(x)} P'} \qquad \frac{P \xrightarrow{\overline{\alpha}(x)} P' \quad P \rhd \alpha \prec \beta}{P \xrightarrow{\overline{\beta}(x)} P'} \qquad \begin{array}{l} a \prec \{b\} = a \curlyvee b \\ \{a\} \prec \{b\} = b \prec a \\ \{a\} \prec b = \text{undefined} \end{array}$$

$$\frac{P \xrightarrow{\mu} P' \qquad a \notin \mathsf{fn}(\mu) \cup \mathsf{bn}(\mu)}{(\nu a)P \xrightarrow{\mu} (\nu a)P'} \qquad \frac{P \xrightarrow{\mu} P' \qquad \mathsf{bn}(\mu) \cap \mathsf{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\mu} P' \mid Q} \qquad \frac{\pi_i.P_i \xrightarrow{\mu} P'}{\Sigma_i \pi_i.P_i \xrightarrow{\mu} P'}$$

Fig. 1. LTS for $\pi\mathtt{P}$. Symmetric versions of the two rules involving $\mid$ are omitted.

without the need to move restrictions. We have:

$$\begin{array}{rl} \overline{a}(c).P \mid a(y).Q \xrightarrow{\tau} & (\nu x)((\nu c)(c/x \mid P) \mid (\nu y)(x/y \mid Q)) \\ \simeq & (\nu c)(\nu y)(P \mid c/y \mid Q) \ . \end{array}$$

3.2 The Characterisation Theorem

Lemma 6. *If $P \equiv Q$ and $P \rhd \varphi$ then $Q \rhd \varphi$.*

Definition 7. *We define a relation $\sqsubseteq^{\varphi}$ between labels as follows: (i) $\alpha_1(x) \sqsubseteq^{\varphi} \alpha_2(x)$ and $\overline{\alpha}_1(x) \sqsubseteq^{\varphi} \overline{\alpha}_2(x)$ when $\varphi = \alpha_2 \prec \alpha_1$, and (ii) $[\varphi_1]\tau \sqsubseteq^{\varphi} [\varphi_2]\tau$ when $\varphi_1, \varphi \vdash \varphi_2$. We write $\sqsubseteq_P$ for the smallest preorder containing all $\sqsubseteq^{\varphi}$ when $P \rhd \varphi$.*

Intuitively, $\eta \sqsubseteq_P \mu$ means that label μ is less general than η, given some condition (φ above) entailed by P. For instance, we have $\{a\}(x) \sqsubseteq_0 a(x)$. This notion is used in the following lemma to reason about transitions of processes.

Lemma 8. *If $P \xrightarrow{\mu} P'$ and $\eta \sqsubseteq_P \mu$ then $P \xrightarrow{\eta} P'$. Conversely, whenever $P \xrightarrow{\eta} P'$, there exists μ such that $\eta \sqsubseteq_P \mu$ and $P \xrightarrow{\mu} P'$, of which there is a proof, not bigger than the one for $P \xrightarrow{\eta} P'$, that does not end with a preorder rule.*

Congruence for parallel composition is proved using Lemma 8, which gives:

Lemma 9. *Relation $\sim$ is a congruence.*

Theorem 10 (Characterisation). *$P \simeq Q$ iff $P \sim Q$.*

Proof (Sketch). The proof follows a standard pattern: soundness is a consequence of Lemma 9. For completeness, we have to show that contexts can express the conditions in the three clauses of Definition 4, and we define accordingly tester

processes. The first clause about φ is handled using process $\overline{\alpha}.\overline{w_1} \mid \beta.\overline{w_2}$ where α and β are such that $\varphi = \alpha \curlyvee \beta$. For transitions (second clause), the counterpart of, e.g., $\xrightarrow{\{a\}(x)}$, is given by tester process $\overline{a}(y).(z/y \mid \overline{w} \mid w)$. We use process φ for the third clause, since $P \xrightarrow{[\varphi]\tau} Q$ iff $P \mid \varphi \xrightarrow{\tau} Q \mid \varphi$. □

As mentioned above, the calculus in [10] is a version of $\pi\mathtt{P}$ with prefixes for free input and output, and without the corresponding bound prefixes. Let us call that calculus $\pi\mathtt{P}_1$. The encoding $[\cdot]_{\mathrm{f}}$, which we introduced in Remark 3, allows us to embed $\pi\mathtt{P}_1$ into $\pi\mathtt{P}$ in a faithful way:

Proposition 11. *$P \simeq_{\pi\mathtt{P}_1} Q$ (in $\pi\mathtt{P}_1$) iff $[P]_{\mathrm{f}} \simeq [Q]_{\mathrm{f}}$ (in $\pi\mathtt{P}$).*

The proof of the above result exploits in a crucial way the fact that, although $\pi\mathtt{P}_1$ does not feature sums and the $[\varphi]\tau$ prefix, those are not needed to prove the completeness of $\sim$.

4 Axiomatisation

4.1 Equational Laws for Strong Bisimilarity

Notations and Terminology. We use A to range over processes that consist of compositions of φ processes only, which we call *preorder processes*. We often view such processes as multisets of conditions. We use notation A, P to denote a process that can be written, *using the monoid laws for parallel composition*, as $A \mid P$, where P does not contain toplevel arcs. Note that A may contain restrictions, but only those corresponding to the definition of join processes (given in Section 2).

We write $\vdash P = Q$ whenever P and Q can be related by equational reasoning using the laws of Figure 2. We omit the standard laws expressing that $\mid$ and $+$ obey the laws of commutative monoids, and that $+$ is idempotent. We also omit the laws for equational reasoning (equivalence, substitutivity). We will reason up to these laws in the remainder.

Comments on the laws. Before presenting the properties of the axiomatisation, we comment on the laws of Figure 2 and illustrate them on some examples.

As usual, expansion (L1) allows us to rewrite the parallel composition of two sum processes into a sum, the third summand describing synchronisation in $\pi\mathtt{P}$.

Preorders. Laws L2-L5 express basic properties of relations $\prec$ and $\curlyvee$, and actually provide an axiomatisation of $\sim$ for preorder processes.

Prefixes. Law L6 propagates φs in depth, expressing the persistence of condition processes (φ). Law L7 is the counterpart of the third clause of Definition 4, and describes the outcome of a $[\varphi]\tau$ transition. Similarly, laws L18-L19 correspond to the firing of visible transitions in the LTS (regarding these rules, see also the comments after Proposition 16).

Expansion law (we can suppose $x \neq y$, $\mathsf{bn}(\pi_i) \notin \mathsf{fn}(T)$, $\mathsf{bn}(\rho_j) \notin \mathsf{fn}(S)$.)

L1 $\underbrace{\sum_i \pi_i.P_i}_{S} \mid \underbrace{\sum_j \rho_j.R_j}_{T} = \sum_i \pi_i.(P_i \mid T) + \sum_j \rho_j.(S \mid R_j) + \sum_{i,j} [\alpha \curlyvee \beta]\tau.(\nu xy)(x/y \mid P_i \mid R_j)$ when $\alpha \curlyvee \beta$ is defined. and $\{\pi_i, \rho_j\} = \{\overline{\alpha}(x), \beta(y)\}$

Laws for preorder processes

L2 $a \prec b \mid b \prec c = a \prec b \mid b \prec c \mid a \prec c$ L3 $a \prec b \mid c \prec b = a \prec b \mid c \prec b \mid a \curlyvee c$

L4 $a \prec b \mid b \curlyvee c = a \prec b \mid b \curlyvee c \mid a \curlyvee c$ L5 $a \prec a = 0$

Laws for prefixes (the counterparts of laws L11-L13 for output are omitted)

L6 $\varphi, S + \pi.P = \varphi, S + \pi.(\varphi \mid P)$ L7 $[\varphi]\tau.P = [\varphi]\tau.(\varphi \mid P)$

L8 $[a \prec a]\tau.P = [b \curlyvee b]\tau.P$

L9 $[a \curlyvee b]\tau.P = [a \curlyvee b]\tau.P + [a \prec b]\tau.P$

L10 $[a \curlyvee b]\tau.P = [a \curlyvee b]\tau.P + [b \curlyvee a]\tau.P$

L11 $a(x).P = a(x).P + \{a\}(x).P$

L12 $b/a, S + a(x).P = b/a, S + a(x).P + b(x).P$

L13 $a/b, S + \{a\}(x).P = a/b, S + \{a\}(x).P + \{b\}(x).P$

L14 $b/a, S + [a \prec c]\tau.P = b/a, S + [a \prec c]\tau.P + [b \prec c]\tau.P$

L15 $a/b, S + [c \prec a]\tau.P = a/b, S + [c \prec a]\tau.P + [c \prec b]\tau.P$

L16 $b/a, S + [a \curlyvee c]\tau.P = b/a, S + [a \curlyvee c]\tau.P + [b \curlyvee c]\tau.P$

L17 $b/a, S + [a \curlyvee c]\tau.P = b/a, S + [a \curlyvee c]\tau.P + [c \prec b]\tau.P$

L18 $\alpha(y).P = \alpha(x).(\nu y)(x/y \mid P)$ if $x \notin \mathsf{fn}(P)$

L19 $\overline{\alpha}(y).P = \overline{\alpha}(x).(\nu y)(y/x \mid P)$ if $x \notin \mathsf{fn}(P)$

Laws for restriction (the counterparts of laws L26 and L27 for output are omitted; $a \prec b \in A^{\neq}$ stands for $a \prec b \in A$ and $a \neq b$, and similarly for $a \curlyvee b$.)

L20 $(\nu b)P = (\nu a)(P\{^a/_b\})$ if $a \notin \mathsf{fn}(P)$ L21 $(\nu c)(\nu d)P = (\nu d)(\nu c)P$

L22 $P \mid (\nu a)Q = (\nu a)(P \mid Q)$ if $a \notin \mathsf{fn}(P)$ L23 $(\nu a)0 = 0$

L24 $(\nu a)A = \{b \prec c \mid b \prec a,\ a \prec c \in A^{\neq}\} \uplus \{b \curlyvee c \mid b \prec a,\ c \prec a \in A^{\neq}\} \uplus \{b \curlyvee c \mid a \curlyvee c,\ b \prec a \in A^{\neq}\} \uplus \{\varphi \in A \mid a \notin \mathsf{n}(\varphi)\}$

L25 $(\nu a)(A,\ S + \pi.P) = (\nu a)\big(A,\ S + \pi.(\nu a)(A \mid P)\big)$ $a \notin \mathsf{n}(\pi)$

L26 $(\nu a)(A,\ S + a(x).P) = (\nu a)\big(A,\ S + \sum_{a \prec b \in A^{\neq}} b(x).(\nu a)(A \mid P) + \sum_{\substack{b \prec a \in A^{\neq} \\ \vee\, a \curlyvee b \in A^{\neq}}} \{b\}(x).(\nu a)(A \mid P)\big)$

L27 $(\nu a)(A,\ S + \{a\}(x).P) = (\nu a)\big(A,\ S + \sum_{b \prec a \in A^{\neq}} \{b\}(x).(\nu a)(A \mid P)\big)$

L28 $(\nu a)(A,\ S + [a \prec c]\tau.P) = (\nu a)\big(A,\ S + \sum_{a \prec b \in A^{\neq}} [b \prec c]\tau.(\nu a)(A \mid P)\big)$ $a \neq c$

L29 $(\nu a)(A,\ S + [c \prec a]\tau.P) = (\nu a)\big(A,\ S + \sum_{b \prec a \in A^{\neq}} [c \prec b]\tau.(\nu a)(A \mid P)\big)$ $a \neq c$

L30 $(\nu a)(A,\ S + [a \curlyvee c]\tau.P) = (\nu a)\big(A,\ S + \sum_{a \prec b \in A^{\neq}} [b \curlyvee c]\tau.(\nu a)(A \mid P) + \sum_{\substack{b \prec a \in A^{\neq} \\ \vee\, a \curlyvee b \in A^{\neq}}} [c \prec b]\tau.(\nu a)(A \mid P)\big)$ $a \neq c$

Fig. 2. An axiomatisation of $\sim$

α-conversion for input prefixes follows from laws L20 and L18, by deriving the following equalities (and similarly for the other visible prefixes):

$$a(y).P \overset{L18}{=} a(x).(\nu y)(x/y \mid P) \overset{L20}{=} a(x).(\nu y')(x/y' \mid P\{^{y'}/_{y}\}) \overset{L18}{=} a(y').P\{^{y'}/_{y}\}\,.$$

Laws L11-L17 can be used to expand process behaviours using the preorder: arcs can modify the subject of visible prefixes (L11-L13) and the condition in $[\varphi]\tau$ prefixes (L14-L17). Laws L9, L10 and L14-L17 rely on the defining properties of relations $\prec$ and $\curlyvee$. Finally, law L8 is used to equate all reflexive τ prefixes.

Restriction. Laws L20-L23 are standard. The other laws are used to "push" restrictions inside processes. Due to the necessity to handle the preorder component (A), they are rather complex.

Law L24 is used to eliminate a restriction on a name a in a preorder process, by propagating the information expressed by all φs that mention a.

Law L25 is rather self-explanatory, and shows how the A component prevents us from simply pushing the restriction downwards (under prefixes).

Laws L26-L30 describe a kind of "synchronous application" of the prefix laws seen above. For instance, the two summands in law L26 correspond to applications of laws L12-L13: as we push the restriction on a downwards, we make sure that all possible applications of these laws are taken into account.

Intuitively, L24 is applied after laws L25-L30 have been used to erase all prefixes mentioning the restricted name a, pushing the restriction on a inwards.

All in all, the set of laws in Figure 2 is rather lengthy. We make two comments on this. First, it can be remarked that axiomatisations often treat restriction separately, by first focusing on a restriction-free calculus. In $\pi\mathtt{P}$, because of preorder processes, we cannot in general push restrictions on top of sum processes, so the situation is more complex (see also the discussion about [14] in Section 5).

Second, we could have presented the laws in a more compact way, by writing *schemas*. A uniform presentation for laws L8-L17 and L26-L30 is as follows:

$$\frac{\eta \sqsubseteq_A \mu \quad \mu.P \in S}{A, S = A, S + \eta.P} \qquad \frac{a \in \mathsf{fn}(\mu) \quad \forall \eta \sqsubseteq_A \mu \quad a \in \mathsf{fn}(\eta) \vee \exists \rho \;\; \eta \sqsubseteq_A \rho \;\wedge\; \rho.P \in S}{(\nu a)(A, \mu.P + S) = (\nu a)(A, S)}$$

(To remove $\mu.P$ from $\mu.P + S$, the second rule requires that some $\rho.P$ are in S. The second rule can be used to add those summands to S.) We prefer nevertheless to write all rules explicitly, since this is how they are handled in proofs.

Examples of derivable equalities. In the following examples, we sometimes switch silently to notation A, P to ease readability. We also allow ourselves to simplify some reasonings involving prefixes where the object is not important. We explain how the following derivable between $\pi\mathtt{P}$ processes can be derived:

$$(\nu a)(b/a \mid a/c) \;=\; b/c \qquad (\nu a)(S + a(x).P) = (\nu a)S$$
$$(\nu a)(a/b \mid a(x).P) \;=\; \{b\}(x).(\nu a)P \qquad \overline{a}(x).x = \overline{a}(x).\{x\} \qquad a(x).\{x\} = a(x).0$$

The first equality above is established using law L24: before getting rid of the restriction on a, we compute all conditions not involving a that can be deduced from $b/a \mid a/c$. In this case, this is only b/c.

The second equality is a direct consequence of law L26. Law L26 is also used for the third equality: only the second sum in the law is not empty, which gives $(\nu a)(a/b, a(x).P) = (\nu a)(a/b, \{b\}(x).(\nu a)P)$. Then, L22 allows us to restrict the scope of νa, and we can get rid of $(\nu a)a/b$ using law 24, which yields the result.

Another way to see the third equality is to observe that we can derive $a/b, a(x).P = a/b, a(x).P + \{a\}(x).P + \{b\}(x).P$ using laws L11 and L13. In the latter process, the sum is intuitively *expanded*, in the sense that all derivable toplevel summands have been made explicit. When considering the restricted version of both processes, it is sound to push the restriction on a downwards in the expanded process, to obtain the expected equality. In this sense, law L26 implements a "synchronous version" of this reasoning, so as to insure that when pushing a restriction downwards, the behaviour of the process is fully expanded.

The next two equalities illustrate the meaning of protected names. We reason as follows: $\overline{a}(x).x \overset{L19}{=} \overline{a}(x').(\nu x)(x/x', x) \overset{L26}{=} \overline{a}(x').(\nu x)(x/x', \{x'\}.(\nu x)(x/x' \mid 0))$. We then obtain the expected equality by getting rid of $(\nu x)x/x'$, twice, using laws L22 and L24. The reason why this equality holds is that fresh name x is emitted without the context having the ability to interact at x, since x will never be under another name in an arc. Therefore, the input at x is equivalent to a protected input.

In the last equality, because of the transition $a(x).\{x\} \xrightarrow{a(x')} (\nu x)(x'/x \mid \{x\})$, x will never be above another name, so that the prefix $\{x\}$ cannot be triggered, and is equivalent to 0. This equality is derived as follows:

$$a(x).\{x\} \overset{L18}{=} a(x').(\nu x)(x'/x \mid \{x\}) \overset{L27}{=} a(x').(\nu x)x'/x \overset{L24}{=} a(x').0 \ .$$

(we have explained above how $a(x').0 = a(x).0$ can be derived).

We leave it to the reader to check that the law for interleaving, presented in Section 1, can be derived using the expansion law, followed by the rules for prefixes and restriction to get rid of the summand $[x \curlyvee y]\tau.(\nu t, u)(t/u)$.

4.2 Soundness and Completeness of the Axioms

Lemma 12 (Soundness). *The laws of Figure 2 relate bisimilar processes.*

Proof (Sketch). For laws 24-30, we establish a "saturation property", expressing the fact that when erasing a preorder process φ or a prefix π that mentions a, we generate all processes φ or π could induce. The other laws are easy. □

Auxiliary Results: Preorder Processes, Prefixes, Restriction.

In order to establish completeness, we first need some technical results, given by Propositions 13, 16 and 17.

First, laws L2-L5 can be used to *saturate* preorder processes:

Proposition 13. *If $A_1, S_1 \sim A_2, S_2$, then there exists $A^\star$ such that $\vdash A_i, S_i = A^\star, S_i$ $(i = 1, 2)$, and $A^\star = \prod\{\varphi \mid \varphi$ not reflexive and $A_1 \rhd \varphi\}$.*

(Note that we could have picked A_2 instead of A_1 above.) We say that A is a *saturated preorder process* whenever $A^\star \equiv A$. We use $A^\star$ to range over such processes. We can remark that even if A contains only arcs, $A^\star$ may contain restrictions, because of induced conditions involving Υ.

The next lemma relates transitions of sum processes and the laws for prefixes.

Lemma 14. *If $A, S \xrightarrow{\mu} A, P$ then $\vdash A, S = A, S + \pi.Q$ for some π and Q such that μ and π only differ in their bound names and $\pi.Q \xrightarrow{\mu} P$.*

Laws L9-L17 can be used to "saturate" the topmost prefixes in sums. We express this using the equivalence below, and rely on Lemma 14 to prove Prop. 16:

Definition 15 (Head sum normal form, $\asymp_h$). *Given two sum processes S and T, we write $S \prec_h T$ whenever for any summand $\pi.P$ of S, there exists a summand $\pi.Q$ of T with $\pi.P \sim \pi.Q$. We let $S \asymp_h T$ stand for $S \prec_h T \wedge T \prec_h S$.*

Proposition 16. *Whenever $A^\star, S_1 \sim A^\star, S_2$, where S_1, S_2 are two sum processes, there are S_1', S_2' s.t. $\vdash A^\star, S_i = A^\star, S_i'$ (for $i = 1, 2$) and $S_1' \asymp_h S_2'$.*

In the definition of $\prec_h$, we impose $\pi.P \sim \pi.Q$, and not simply $P \sim Q$. The equivalence induced by the choice of the latter condition would indeed be too discriminating. To see why, consider $Q_1 = a(x).c\not/x$ and $Q_2 = a(x).0$. Obviously, $c\not/x \not\sim 0$. On the other hand, we have $Q_1 \sim Q_2$: after a $\xrightarrow{a(y)}$ transition on both sides, we must compare $(\nu x)(c\not/x \mid y/x)$ and $(\nu x)(y/x)$, and both are bisimilar to 0. In order to derive $\vdash Q_1 = Q_2$, we rely on the following property, which explains the shape of laws L18, L19: $a(y).P \sim a(y).Q$ iff $(\nu y)(x/y \mid P) \sim (\nu y)(x/y \mid Q)$.

Proposition 17 expresses that restrictions can be pushed inwards in processes. It introduces a notion of measure on processes that is useful to reason by induction on processes in the completeness proof:

Proposition 17. *We define $|P|$ as follows: $|\Sigma_i \pi_i.P_i| = \max_i (1 + |P_i|)$ $|(\nu a)P| = |P|$, $|P \mid Q| = |P| + |Q|$, and $|a/b| = 0$.*

For any A, S, a, there exist A' and S' such that $\vdash (\nu a)(A, S) = A', S'$ and $|(\nu a)(A, S)| \geq |A', S'|$.

Establishing Completeness. The grammar $P ::= A, \Sigma_i \pi_i.P_i \mid (\nu a)P$ defines what we call *$|$-free processes*: only arcs are composed, and the non-preorder part of processes is a sum.

Proposition 18. *For all $|$-free processes P and Q, $P \sim Q$ iff $\vdash P = Q$.*

Proof (Sketch). The 'if' part follows from Lemmas 9 and 12. Suppose now $P \sim Q$; we reason by induction on $|P| + |Q|$. By Propositions 17, 13 and 16, we obtain $\vdash P = A^\star, S_1$ and $\vdash Q = A^\star, S_2$, for some A, S_1, S_2 such that $S_1 \asymp_h S_2$.

We then consider $a(x).T_1 \in S_1$ and $a(x).T_2 \in S_2$ s.t. $a(x).T_1 \sim a(x).T_2$. The latter yields, by triggering the input transition, $(\nu x)(y/x \mid T_1) \sim (\nu x)(y/x \mid T_2)$. By induction we derive $\vdash (\nu x)(y/x \mid T_1) = (\nu x)(y/x \mid T_2)$ from which we get $\vdash a(x).T_1 = a(x).T_2$ by law L18.

The other kinds of prefixes are handled similarly. This reasoning allows us to prove $\vdash S_1 = S_2$ and hence $\vdash P = Q$. □

The expansion law yields the following result, which then gives Theorem 20.

Lemma 19. *For any P, there exists a $|$-free process Q s.t. $\vdash P = Q$.*

Theorem 20 (Axiomatisation of $\sim$). *For all P and Q, $P \sim Q$ iff $\vdash P = Q$.*

Remark 21 (Normal forms). The proofs of the results in this section suggest that we can define a strategy to apply the rules of Figure 2, in order to rewrite a $\pi\mathtt{P}$ process P to its *normal form*, $\mathsf{nf}(P)$, so that $P \sim Q$ iff $\mathsf{nf}(P) = \mathsf{nf}(Q)$. We leave the rigorous description of this normalisation procedure for future work.

4.3 Adapting our Axiomatisation to Explicit Fusions

We can reuse the ideas presented above to describe an axiomatisation for barbed congruence in Explicit Fusions (EF, [20]). EF feature *fusion processes*, of the form $a{=}b$, which can equate names via $\equiv$: we have $a{=}b \mid P \equiv a{=}b \mid P\{b/a\}$.

Like in $\pi\mathtt{P}$, we work with three kinds of prefixes, $\overline{a}(x)$, $a(x)$ and $[a{=}b]\tau$, the latter being the counterpart of $[\varphi]\tau$ in $\pi\mathtt{P}$ (it appears, e.g., in [20]).

The "state component" of processes is simpler in EF than in $\pi\mathtt{P}$, since fusions implement an equivalence relation on names. The laws of Figure 2 can be ported to EF, yielding an axiomatisation. We only discuss some relevant laws, and refer to [11] for a complete definition. The laws for fusion processes are

$$a{=}a = 0 \qquad a{=}b \mid a{=}c = a{=}b \mid a{=}c \mid b{=}c \qquad a{=}b = b{=}a \ ,$$

and the EF counterpart of (some of) the laws for prefixes in $\pi\mathtt{P}$ is given by

$$a{=}b \mid a(x).P = a{=}b \mid b(x).P \qquad a{=}b \mid [a{=}c]\tau.P = a{=}b \mid [b{=}c]\tau.P$$

(laws L6-L7 are inherited directly, φ denoting fusions). Because fusions satisfy transitivity, every fusion can be eliminated if one of its names is restricted, as $(\nu a)(a{=}b \mid P) \sim P\{{}^{b}/_{a}\}$. This makes the laws for restriction much simpler than in $\pi\mathtt{P}$:

$$(\nu a)a{=}b = 0 \qquad (\nu a)\Sigma_i \pi_i.P_i = \Sigma_{i|a \notin \mathsf{n}(\pi_i)} \pi_i.(\nu a)P_i \ .$$

5 Conclusions and Future Work

Working with a preorder on names has an influence on the behavioural theory of $\pi\mathtt{P}$, notably through the interplay between arcs and restrictions. The preorder relation is represented explicitly in $\pi\mathtt{P}$ processes, using arcs. We do not see any natural "implicit version" of $\pi\mathtt{P}$, mimicking the relation between Explicit Fusions and Fusions, whereby the extension of the preorder along a communication would not generate an arc process.

The stateful nature of the preorder component of $\pi\mathtt{P}$ processes can be related to *frames* in the applied π-calculus [1] and Psi-calculi [2]. Arcs in $\pi\mathtt{P}$ can be seen in some sense as substitutions, but they differ from the active substitutions of applied π. The latter map variables to terms, while, in the tradition of fusion calculi, we only have (channel) names in $\pi\mathtt{P}$. Moreover, several arcs acting on the same name are allowed in $\pi\mathtt{P}$, while a substitution acts on at most one variable in applied π. For these reasons, the behavioural theories of $\pi\mathtt{P}$ and applied π are rather different. Liu and Lin's proof system for applied π [14] departs from our axiomatisation for $\pi\mathtt{P}$, but has in common the stateful component of processes.

Psi-calculi can represent the active substitutions of applied π. It would be interesting to study whether arcs, and the preorder between names, can be represented in the setting of Psi-calculi. An important technical point to address in this perspective is whether transitivity of (generalised) channel equivalence in Psi-calculi would conflict with the fact that name joinability is not transitive in $\pi\mathtt{P}$. Another important feature of Psi-calculi is that they come with a fully mechanised metatheory: this is clearly something that $\pi\mathtt{P}$ is lacking at the moment.

The behavioural theory of $\pi\mathtt{P}$ is based on an operational account. An intriguing question is the construction of a denotational model for $\pi\mathtt{P}$, and the comparison with known models for π and Fusions. We would also like to study the weak version of behavioural equivalence.

The results of this work provide foundations for the behavioural theory of the $\pi\mathtt{P}$ calculus, which also has i/o-types (cf. [10]). As already mentioned, typed behavioural equivalence [17,6] can be used to establish fine behavioural properties of concurrent systems. We would like to find out whether it can be helpful to refine untyped analyses of systems where Fusions have been used.

Acknowledgements. We thank Davide Sangiorgi and Fu Yuxi for useful discussions about this work. This work has been supported by projects ANR 12IS02001 PACE, ANR 2010-BLAN-0305 PiCoq and NSF of China (61261130589).

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of POPL*, pages 104–115. ACM, 2001.
2. J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *LICS*, page 39–48. IEEE, 2009.
3. M. Boreale, M. G. Buscemi, and U. Montanari. D-fusion: A distinctive fusion calculus. In *Proc. APLAS*, volume 3302 of *LNCS*, pages 296–310. Springer, 2004.
4. M. Boreale, M. G. Buscemi, and U. Montanari. A general name binding mechanism. In *Proc. TGC*, volume 3705 of *LNCS*, pages 61–74. Springer, 2005.
5. S. Carpineti, C. Laneve, and L. Padovani. PiDuce - A project for experimenting Web services technologies. *Sci. Comput. Program.*, 74(10):777–811, 2009.
6. Y. Deng and D. Sangiorgi. Towards an algebraic theory of typed mobile processes. *Theor. Comput. Sci.*, 350(2-3):188–212, 2006.
7. T. Ehrhard and O. Laurent. Acyclic solos and differential interaction nets. *Logical Methods in Computer Science*, 6(3), 2010.
8. Y. Fu. The χ-calculus. In *APDC*, pages 74–81. IEEE Computer Society, 1997.

9. P. Gardner, C. Laneve, and L. Wischik. The fusion machine. In *CONCUR*, volume 2421 of *Lecture Notes in Computer Science*, pages 418–433. Springer, 2002.
10. D. Hirschkoff, J.-M. Madiot, and D. Sangiorgi. Name-passing calculi: From fusions to preorders and types. In *LICS*, pages 378–387. IEEE Computer Society, 2013.
11. D. Hirschkoff, J.-M. Madiot, and X. Xu. Long version of this paper. Available from `http://madiot.org`.
12. K. Honda and N. Yoshida. On reduction-based process semantics. *Theor. Comp. Sci.*, 152(2):437–486, 1995.
13. C. Laneve and B. Victor. Solos in concert. *Mathematical Structures in Computer Science*, 13(5):657–683, 2003.
14. J. Liu and H. Lin. Proof system for applied pi calculus. In *Proc. IFIP TCS*, volume 323 of *IFIP Advances in Inf. and Comm. Technol.*, pages 229–243. Springer, 2010.
15. J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. *Inf. Comput.*, 120(2):174–197, 1995.
16. J. Parrow and B. Victor. The fusion calculus: expressiveness and symmetry in mobile processes. In *LICS*, pages 176 –185. IEEE, 1998.
17. B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.
18. D. Sangiorgi and D. Walker. *The Pi-Calculus: a theory of mobile processes*. Cambridge University Press, 2001.
19. B. Victor and J. Parrow. Concurrent constraints in the fusion calculus. In *Proc. ICALP*, volume 1443, pages 455–469, 1998.
20. L. Wischik and P. Gardner. Strong bisimulation for the explicit fusion calculus. In *Proc. of FoSSaCS*, volume 2987 of *LNCS*, pages 484–498, 2004.