A Hierarchy of Equivalences for Asynchronous Calculi ⋆

Cédric Fournet [a], Georges Gonthier [b]

[a] *Microsoft Research, 7 J J Thomson Avenue, Cambridge CB3 0FB, UK*

[b] *INRIA Rocquencourt, BP 105, 78153 Le Chesnay, France*

Abstract

We generate a natural hierarchy of equivalences for asynchronous name-passing process calculi from simple variations on Milner and Sangiorgi's definition of weak barbed bisimulation. The π-calculus, used here, and the join calculus are examples of such calculi.

We prove that barbed congruence coincides with Honda and Yoshida's reduction equivalence, and with asynchronous labeled bisimulation when the calculus includes name matching, thus closing those two conjectures.

We also show that barbed congruence is coarser when only one barb is tested. For the π-calculus, it becomes a limit bisimulation, whereas for the join calculus, it coincides with both fair testing equivalence and with the weak barbed version of Sjödin and Parrow's coupled simulation.

⋆ A preliminary extended abstract appeared in [16]

To appear in JLAP special issue on the Pi-Calculus *10/2001 (revised:7/2003)*

Contents

1 Introduction

There is a large number of proposals for the "right" equivalence for concurrent processes—see for instance van Glabbeek's impressive overview of weak equivalences [20]. Choosing the proper equivalence to state a correctness argument often means striking a delicate balance between intuitively compelling statements and manageable proof techniques. For instance, there are many effective, sometimes automated techniques for proving bisimulation-based equivalences, even for infinite systems, but it can be quite hard to prove that two processes are *not* bisimilar—and to interpret this situation—because bisimulation does not directly correspond to an operational model. On the other hand, the proof that two processes are not testing equivalent is simply a failure scenario, but it can be quite hard to directly prove a testing equivalence.

In this paper, we cast some of these equivalences in a simple unifying hierarchy—summarized in Figures 3–5 of Section 9. While the equivalences are hardly new, our results relate different styles of definition: trace-based versus bisimulation-based, labeled semantics versus reduction semantics, fairness versus coupled simulations, limit bisimulations versus co-inductive bisimulations. We identify four main equivalences, with increasing discriminating power. In this hierarchy, one can start a proof effort at the upper tier with a simple labeled bisimulation proof; if this fails, one can switch to a coarser equivalence by augmenting the partial proof; if the proof still fails for the testing equivalences in the last tiers, then meaningful counter-examples can be found. The hierarchy is backed by several new results:

- We close a conjecture of Honda and Yoshida [23] by showing that barbed equivalence equals their reduction-based equivalence, with or without name matching (Theorem 1).
- We close a conjecture of Milner and Sangiorgi [32] by showing that labeled bisimilarity equals barbed equivalence for all processes in the presence of name matching (Theorem 5).
- We show that barbed equivalence with a single test is strictly coarser than the corresponding reduction-based equivalence. In the π-calculus, it yields a surprising limit bisimulation (Theorem 2). In the join calculus, or in the π-calculus with an adapted definition of observation, it yields fair testing equivalence (Theorem 3).
- We bridge the gap between bisimulation and testing equivalences by showing that fair testing [9,34,10] coincides with a form of coupled simulation [37] (Theorem 4).
- Conversely, we provide counter-examples that establish several strict inclusions between equivalence relations.

Before discussing these technical subtleties, we spend some time to sketch a general picture and to motivate our choices. Our framework is based on abstract reduction systems $(\mathcal{P}, \rightarrow, \downarrow_x)$, where $\mathcal{P}$ is a set of processes, $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$ is a reduction

relation on processes, and $\downarrow_x$ is a family of predicates on processes. The predicates $\downarrow_x$ are syntactic properties meant to detect the outcome of the computation (e.g., "success", convergence,...). This style of definition is relatively independent of syntactic details, is adapted for higher-order settings, and is especially convenient to relate different calculi. The most studied reduction system is probably the λ-calculus. In process calculi based on labeled transition systems, such as CCS or the π-calculus, the reductions are the internal (τ) transitions and the predicates are immediate communication capabilities—the barbs [32]. These predicates induce equivalences and preorders on processes, which can then be refined by additional requirements such as context-closure or bisimulation.

We are interested in equivalences for asynchronous concurrent systems. This motivates our choice of equivalences, exclusively defined in terms of weak reductions ($\rightarrow^*$) and weak barbs ($\rightarrow^*\downarrow_x$). However, many results on those equivalences do not depend on asynchrony. Although our results were first obtained in the join calculus, they are stated here in the more familiar asynchronous π-calculus [6], which enjoys similar properties in this respect. (The main exceptions are discussed in Section 6.) Some inclusions between equivalences are general and easily established; others are less immediate and more specific to the π-calculus; their proofs typically rely on some encoding.

The paper is organized as follows. In Section 2, we review the syntax, operational semantics, and types for the asynchronous π-calculus. In Section 3, we define evaluation contexts and barbs, introduce two basic equivalences, may testing and barbed congruence, and discuss context-closure properties. In Section 4, we study intermediate equivalences, fair testing and barbed coupled congruence. In Sections 5 and 6, we reconsider our choice of observations: we define *existential tests* and *committed tests*, respectively, and explore the resulting variants for all our equivalences. In Section 7, we focus on an auxiliary notion of equivalence, doubled-barbed bisimilarity, and use it to prove Theorem 1. In Section 8, we finally consider labeled semantics. In Section 9, we summarize our results as a hierarchy of equivalences, for reduction systems in general and for the asynchronous π-calculus in particular.

Notations We write $\widetilde{t}$ for a tuple of terms $t_1, \ldots, t_n$ of length $n \geq 0$. All our relations are binary. We usually adopt an infix notation for relations. We write Id for the identity relation. Let $\mathcal{R}$ and $\mathcal{R}'$ be two relations. We write $\mathcal{R}\mathcal{R}'$ for the composition of relations $\{(x,y) \mid \exists z. x\ \mathcal{R}\ z\ \mathcal{R}'\ y\}$, $\mathcal{R}^{-1}$ for the converse relation $\{(y,x) \mid x\ \mathcal{R}\ y\}$, $\mathcal{R}^n$ for the repeated relation inductively defined by $\mathcal{R}^0 = Id$ and $\mathcal{R}^{n+1} = \mathcal{R}\mathcal{R}^n$, $\mathcal{R}^=$ for the reflexive closure $Id \cup \mathcal{R}$, and $\mathcal{R}^*$ for the reflexive-transitive closure $\bigcup_{n\geq 0} \mathcal{R}^n$. We usually adapt postfix notations for predicates. Every relation $\mathcal{R}$ defines an existential predicate, also written $\mathcal{R}$, defined by $x\mathcal{R} = \exists y \mid x\,\mathcal{R}\,y$. Let $\downarrow$ be a predicate; the relation $\mathcal{R}$ *refines* $\downarrow$ when for all P, Q such that $P\ \mathcal{R}\ Q$, $P\downarrow$ implies $Q\downarrow$.

2 An Asynchronous π-calculus (Review)

In this paper, we focus on a core, polyadic, asynchronous π-calculus. We assume some knowledge of the π-calculus, and refer to [30,31,43] for more details and explanations. Our notations and definitions are mostly standard. We use the following grammar for processes:

$P, Q, R ::=$		processes
	$\overline{x}\langle z_1, \ldots, z_n\rangle$	asynchronous emission
$\mid$	$x(y_1, \ldots, y_n).Q$	reception
$\mid$	$\mathsf{0}$	null process
$\mid$	$P \mid P'$	parallel composition
$\mid$	$!Q$	replication
$\mid$	$\nu y.P$	scope restriction
$\mid$	$[x = z]Q$	name matching (optional)

where the names $y_1, \ldots, y_n$ are pairwise distinct. We say that a process is *guarded* when it occurs under a reception, a replication, or a name matching (processes Q above). We use the following abbreviations for processes: $\overline{x}$ for $\overline{x}\langle\rangle$, $x.P$ for $x().P$, $x(\widetilde{y})$ for $x(\widetilde{y}).\mathsf{0}$, and $\nu y_1, \ldots, y_n.P$ for $\nu y_1.\ldots.\nu y_n.P$.

We assume given a countable set of names $x, y, z, \ldots \in \mathcal{N}$. Names appearing in a process can either be free, or be bound by a reception or a restriction (names $y_1, \ldots, y_n$ and y in the grammar above). We write $\mathrm{fv}(P)$ for the free names of P. As for λ-terms, we say that a process has sort S when $\mathrm{fv}(P) \subseteq S$.

The operational semantics follows those given in [43]. Structural equivalence, $\equiv$, is the smallest equivalence on processes that meets the equations below and is closed by application of evaluation contexts and renamings of bound names:

$$P \equiv P \mid \mathsf{0} \qquad \nu x.\mathsf{0} \equiv \mathsf{0}$$

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad \nu x.\nu y.P \equiv \nu y.\nu x.P$$

$$P \mid Q \equiv Q \mid P \qquad P \mid \nu x.Q \equiv \nu x.(P \mid Q) \text{ when } x \notin \mathrm{fv}(P)$$

$$!P \equiv P \mid !P$$

Reduction steps $\rightarrow$, input transitions $\xrightarrow{x(\tilde{y})}$, and output transitions $\xrightarrow{(\tilde{z})\overline{x}\langle\tilde{y}\rangle}$ are the smallest relations on processes that meet the equations below. (We use here the asynchronous input rule initially proposed by Honda and Yoshida [23].) In the equations, $\xrightarrow{\alpha}$ ranges over any of these relations, and $\mathrm{fv}(\alpha)$ and $\mathrm{bv}(\alpha)$ are the free names and bound names of α, respectively.

$$\overline{x}\langle\widetilde{y}\rangle \mid x(\widetilde{z}).Q \;\to\; Q\{\widetilde{y}/\widetilde{z}\}$$

$$[x = x]Q \;\to\; Q$$

$$\mathsf{0} \xrightarrow{x(\widetilde{y})} \overline{x}\langle\widetilde{y}\rangle$$

$$\overline{x}\langle\widetilde{y}\rangle \xrightarrow{\overline{x}\langle\widetilde{y}\rangle} \mathsf{0}$$

$$\frac{P \equiv \xrightarrow{\alpha} \equiv Q}{P \xrightarrow{\alpha} Q}$$

$$\frac{P \xrightarrow{\alpha} Q \qquad \mathrm{fv}(R) \cap \mathrm{bv}(\alpha) = \emptyset}{P|R \xrightarrow{\alpha} Q|R}$$

$$\frac{P \xrightarrow{\alpha} Q \qquad x \notin \mathrm{fv}(\alpha) \cup \mathrm{bv}(\alpha)}{\nu x.P \xrightarrow{\alpha} \nu x.Q}$$

$$\frac{P \xrightarrow{(\widetilde{z})\overline{x}\langle\widetilde{y}\rangle} Q \qquad t \in \mathrm{fv}((\widetilde{z})\overline{x}\langle\widetilde{y}\rangle) \setminus \{x\}}{\nu t.P \xrightarrow{(t,\widetilde{z})\overline{x}\langle\widetilde{y}\rangle} Q}$$

We will need name matching and labeled transitions only in Sections 7 and 8. By default, we always consider processes and contexts without the name matching prefix. Otherwise, we explicitly mention terms in the "π-calculus with matching".

Although the presence of a type system is usually irrelevant, some encodings depend on its expressiveness. We rely on the (simple, recursive, pure) type system given in [43, sections 6.4–6.7]. We use the following grammar for the types of names:

$\sigma, \tau ::=$		communication types
	$\langle\sigma_1, \ldots, \sigma_n\rangle$	channel type
$\mid$	$\alpha, o, \ldots$	type variable
$\mid$	$\mu o.\sigma$	recursive type

We identify types that are equal by renaming of μ-bound variables, by folding, and by unfolding of recursive types. We always assume that our terms are well-typed, even though we usually omit type annotations; when we need to be explicit (e.g., in Lemma 36), we only annotate the scope restriction construct, as in [43]. Likewise, we usually keep the typing context implicit. We say that a name is nullary when it has type $\langle\rangle$ in this implicit context.

Our calculus does not have a primitive choice operator, or a silent action. Instead, we define a derived *internal choice* operator $\bigoplus_{i\in I} P_i \stackrel{\text{def}}{=} \nu t.\, (\bar{t} \mid \prod_i t.P_i)$ where I is a finite set and t is a name that does not appear in any P_i. We write $P_1 \oplus \cdots \oplus P_n$ for $\bigoplus_{i=1\ldots n} P_i$, and write $\tau.P_1$ for $\bigoplus_{i=1} P_1$. More generally, we say that $P = \bigoplus_{P_i\in\mathcal{P}} P_i$ is an internal choice on $\mathcal{P}$ for some equivalence ϕ when:

(1) for all $P_i \in \mathcal{P}$, we have $P \to^* \phi\ P_i$.
(2) if $P \to^*\ P'$, then either P' is an internal choice on $\mathcal{P}$ with $P\ \phi\ P'$, or there exists $P_i \in \mathcal{P}$ with $P_i \to^* \phi\ P'$.
(3) P does not communicate on free names.

For finite $\mathcal{P}$, with the implementation above, this property holds for strong labeled bisimilarity.

3 Congruences, Tests, and Bisimulations

In order to define observational equivalences, we first set up notions of context closure and basic observation. As usual, contexts are processes with a hole, written $C[\ ]$. For some given family of contexts, and to every relation ϕ on processes, we associate its congruence closure $\phi^\circ \stackrel{\text{def}}{=} \{(P,Q) \mid \forall C[\ ].C[P]\ \phi\ C[Q]\}$. A relation (resp. an equivalence) ϕ is a precongruence (resp. a congruence) when $\phi = \phi^\circ$.

We define our notions of congruence and precongruence for a particular class of contexts: an *evaluation context* is a context where the hole $[\ ]$ occurs exactly once, and not under a guard— these contexts are called static contexts in [29]. Evaluation contexts describe environments that can communicate with the process being observed, but can neither replicate it nor prevent its internal reductions. Since we are mostly interested in congruences for evaluation contexts, we will use plain relation symbols ($\simeq$, $\approx$,...) for them, and dotted relation symbols ($\dot{\simeq}$, $\dot{\approx}$,...) otherwise.

In the π-calculus, evaluation contexts are given by the grammar:

$$C[\ ] ::= [\ ] \parallel Q \,|\, C[\ ] \parallel C[\ ] \,|\, Q \parallel \nu x.C[\ ]$$

In this paper, all context closure properties refer to these contexts. Up to structural equivalence, they are of the form $C[\ ] = \nu\widetilde{y}.([\ ] \,|\, Q)$: for all $C[\ ]$ that bind the names $\widetilde{x}$, and for all processes P, there exist a process Q and distinct names $\widetilde{y}$ such that $C[P] \equiv \nu\widetilde{y}.(P\{\widetilde{y}/\widetilde{x}\} \,|\, Q)$.

Our π-calculus is asynchronous in the sense of [8,6]: in a given process, emission on a free name x can be observed using, for instance, an evaluation context $x(\widetilde{y}).P \,|[\ ]$ with a reception on x that can trigger any process P. Conversely, reception on x is not directly observable using an emission on x, because emissions don't have guarded processes; for example, the process $x.\overline{x}$ is not detectable. We define our observation predicates accordingly:

Definition 1 *The predicate $\downarrow_x$—the strong barb on x—detects whether a process emits on name x in an evaluation context: $P \downarrow_x$ if and only if $P \equiv \nu\widetilde{y}.(\overline{x}\langle\widetilde{z}\rangle \,|\, Q)$ with $x \notin \{\widetilde{y}\}$.*

The barbs only detect the superficial behavior of a process—for instance they do not separate $\overline{x}\langle y\rangle$ from $\overline{x}\langle z\rangle$—but in combination with the congruence property they provide a behavioral account of processes.

3.1 May Testing

Testing semantics have a long history, which can be traced back to the Morris equivalence for the λ-calculus [33]. As regards process calculi, they have been proposed

for CCS in [12,21,29] then extended to the π-calculus [7] and the join calculus [24].

Testing semantics are usually defined as a preorder relation $\sqsubseteq$ (the corresponding equivalence being $\sqsubseteq \cap \sqsubseteq^{-1}$). This preorder is commonly interpreted as the "correct implementation" relation: an implementation can rule out some traces, but not exhibit traces whose behavior is not captured by their specification. This direct interpretation is an advantage of testing equivalences over bisimulations, which are typically strictly finer [29].

In general, a test is an observer plus a way of observing; here, observers are evaluation contexts and observations are barbs:

Definition 2 *The* may predicate $\Downarrow_x$*—the barb on* x*—detects whether a process can emit on* x*, possibly after performing some internal reductions. The* may testing equivalence $\simeq_{\mathrm{may}}$ *(resp. the* may testing preorder $\sqsubseteq_{\mathrm{may}}$*) is the largest congruence (resp. precongruence) that respects the barbs* $\Downarrow_x$*.*

$$P \Downarrow_x \stackrel{def}{=} \exists P'.P \rightarrow^* P' \downarrow_x$$

$$P \sqsubseteq_{\mathrm{may}} Q \stackrel{def}{=} \forall C[\,], x.\ C[P] \Downarrow_x \textit{ implies } C[Q] \Downarrow_x$$
$$P \simeq_{\mathrm{may}} Q \stackrel{def}{=} \forall C[\,], x.\ C[P] \Downarrow_x \textit{ if and only if } C[Q] \Downarrow_x$$

A typical example of may testing equivalence is $P \oplus \mathtt{0} \simeq_{\mathrm{may}} P$ for any process P. May testing is most useful to prove safety properties: the specification of a program says that bad things should never happen. Thus suitable behaviors are characterized as those with no bad barbs. For example, it is adequate to specify security properties in cryptographic protocols [4]. However, may testing says nothing on the presence of suitable behaviors. In Section 4, we consider other testing semantics that address this issue.

3.2 Bisimulations and Congruences

Bisimulation-based equivalences [36,29] are often preferred to testing semantics for the π-calculus. Independently of their theoretical appeal, they can be established by co-induction, by considering only a few reduction steps at a time instead of whole traces. Moreover, numerous sophisticated techniques lead to smaller candidate bisimulations, and to modular proofs (see [41,43] for some examples).

Definition 3 *A relation* $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ *is a* (weak, reduction-based) simulation *if, for all* P, P', Q *such that* $P\ \mathcal{R}\ Q$ *and* $P \rightarrow^* P'$*, there exists* Q' *such that* $Q \rightarrow^* Q'$ *and* $P'\ \mathcal{R}\ Q'$*. In short:* $\mathcal{R}^{-1}{\rightarrow^*} \subseteq {\rightarrow^*}\mathcal{R}^{-1}$*.*

Barbed bisimilarity has been proposed by Milner and Sangiorgi [32] as a uniform basis to define behavioral equivalences on different process calculi:

Definition 4 *A simulation $\mathcal{R}$ is a* barbed simulation *when it refines all barbs: if $P\ \mathcal{R}\ Q$ and $P \Downarrow_x$, then $Q \Downarrow_x$. A relation $\mathcal{R}$ is an barbed bisimulation when both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are barbed simulations. The largest barbed bisimulation is called* barbed bisimilarity, *and is written $\dot{\approx}$.*

This style of definition is not entirely unrelated to testing semantics:

Proposition 5 *In any reduction system $(\mathcal{P}, \rightarrow, \downarrow_x)$, (1) the largest barbed simulation is the preorder that refines all barbs $\Downarrow_x$; (2) its precongruence, the may testing preorder $\sqsubseteq_{\mathrm{may}}$, is the largest precongruence that is a barbed simulation.*

PROOF. (1) The preorder that refines all barbs is a weak simulation, since any reduction steps can be trivially simulated by no step. Conversely, the largest weak simulation is also a preorder.

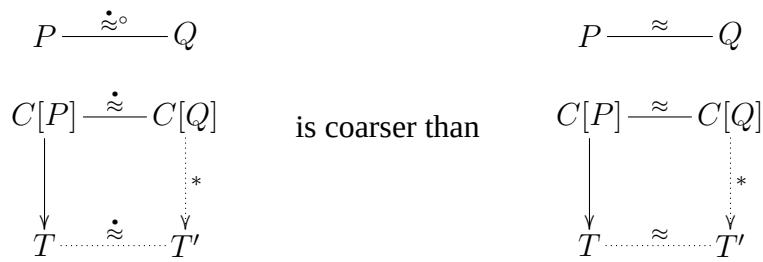
(2) By definition, $\sqsubseteq_{\mathrm{may}}$ is a precongruence; using the first part of the theorem, it is also a barbed simulation, hence it is included in the largest precongruence that is a barbed simulation. The converse inclusion holds by definition. □

Unlike may testing, however, barbed bisimulation reveals the internal branching structure of processes, and thus it induces congruences finer than testing semantics. Remarkably, there are at least two reasonable ways of ensuring the congruence property:

- either take the largest congruence included in the largest barbed bisimulation; this is the two-stage definition traditionally chosen for CCS and the π-calculus, e.g. [32,39,43];
- or take the largest congruence that is a barbed bisimulation; this is essentially the "reduction-based" equivalence chosen for the ν-calculus in [22,23], and the barbed congruence used in our previous works [15,3,2,1].

Definition 6 Barbed equivalence, *written $\dot{\approx}^\circ$, is the largest congruence included in barbed bisimilarity.* Barbed congruence, *written $\approx$, is the largest congruence that is a barbed bisimulation.*

By definition, the two congruences coincide if and only if $\dot{\approx}^\circ$ is itself a bisimulation, but this is not necessarily the case (we give counterexamples in Sections 5, 6, and 7) and in general we only have $\approx \subseteq \dot{\approx}^\circ$. The two diagrams below stress the difference between the two definitions:



(As usual in bisimulation diagrams, we use plain and dotted lines to represent universally- and existentially-quantified relations, respectively.) For processes related by $\dot{\approx}^\circ$, the relation that is preserved in bisimulation diagrams after applying the congruence property is $\dot{\approx}$, and not $\dot{\approx}^\circ$; on the contrary, the congruence property of $\approx$ is preserved through repeated applications of bisimulation and congruence properties.

Technically, the two definitions also induce different kinds of candidate relations in co-inductive proofs. As illustrated in this paper, $\approx$ seems easier to establish than $\dot{\approx}^\circ$. Fortunately, the two equivalences coincide in our setting:

Theorem 1 *In the π-calculus, we have $\dot{\approx}^\circ = \approx$.*

The proof relies on a variant of bisimilarity with two barbs and a series of encodings. It is detailed in Section 7.

To conclude this section, we recall standard but useful properties of barbed congruence. We omit their proofs. We begin with a convenient proof technique (see [43, section 2.4]):

Lemma 7 *To establish $\mathcal{R} \subseteq \approx$, it suffices to show that, for all $P\ \mathcal{R}\ Q$:*

(1) if $P \downarrow_x$, then $Q \Downarrow_x$; conversely if $Q \downarrow_x$, then $P \Downarrow_x$;
(2) if $P \to P'$, then there is Q' such that $Q \to^* Q'$ and $P' \equiv\mathcal{R}^{=}\approx Q'$;
if $Q \to Q'$, then there is P' such that $P \to^* P'$ and $P' \approx\mathcal{R}^{=}\equiv Q'$;
(3) For all evaluation contexts C, we have $C[P] \approx\mathcal{R}^{=}\approx C[Q]$.

The next proposition introduces Honda and Yoshida's "equators" [23], that is, processes that make two names indistinguishable by forwarding any messages sent on one of those names to the other.

Proposition 8 (Equators) *Let $E_x^y \stackrel{def}{=} !x(\widetilde{z}).\overline{y}\langle\widetilde{z}\rangle \,|\, !y(\widetilde{z}).\overline{x}\langle\widetilde{z}\rangle$. For all π-calculus processes P such that $P \mid E_x^y$ is well-typed, we have $\nu x.(P \mid E_x^y) \approx P\{{}^{y}/_{x}\}$.*

The equation above relies on a key property of asynchronous systems: the presence of intermediate buffers on communication channels cannot be observed. As discussed in Section 8, this equation holds only in the absence of name matching.

In our definition of congruence, we consider only evaluation contexts. However, we can systematically use the equation above to obtain stronger context-closure properties for congruences coarser than barbed congruence:

Corollary 9 *In the π-calculus, (1) let ϕ be a precongruence such that $\approx \subseteq \phi$. Then ϕ is also closed by substitutions on free names. (2) The relations $\sqsubseteq_{\mathrm{may}}$, $\dot{\approx}^{\circ}$, and $\approx$ are closed by application of arbitrary π-calculus contexts.*

PROOF. (1) Since any given processes related by ϕ have a finite number of free names, it suffices to prove that ϕ is closed by all single substitutions $\{^{y}/_{x}\}$. If $P \;\phi\; Q$, then $\nu x.(E_x^y \mid P) \;\phi\; \nu x.(E_x^y \mid Q)$ using the precongruence property of ϕ. By Proposition 8, we have $\nu x.(E_x^y \mid P) \;\approx\; P\{^{y}/_{x}\}$, hence $\nu x.(E_x^y \mid Q) \;\phi\; P\{^{y}/_{x}\}$ and, by transitivity, $P\{^{y}/_{x}\} \;\phi\; Q\{^{y}/_{x}\}$.

(2) The proofs are standard; they rely on (1) for the input guards. □

4 Fair Testing and Coupled Simulations

In this section, we attempt to reconcile testing semantics and bisimulation-based semantics by considering intermediate equivalences between $\simeq_{\mathrm{may}}$ and $\approx$.

4.1 Fair Testing

We first consider how may testing can be refined to capture the positive behavior of processes. The usual approach is to observe messages that are always emitted, independently of internal choices: the *must predicate* detects outputs that are present on all finite traces ($P \,\Box{\downarrow_x} \stackrel{\text{def}}{=} \forall P' \;.\; P \rightarrow^* P' \not\rightarrow$ implies $P' \downarrow_x$) and can be used to define must testing and may-and-must testing equivalences as in Definition 2. These relations, however, are not asynchronous, and they are unduly sensitive to diverging behaviors: they interpret all infinite computations in the same manner. Instead, one can modify the must predicate to incorporate a notion of "abstract fairness", and obtain a fine testing equivalence, initially proposed for variants of CCS by Brinksma, Rensink, and Vogler [9,10] and Natarajan and Cleaveland [34].

Definition 10 *The* fair-must predicate $\Box{\Downarrow_x}$ *detects whether a process always retains the possibility of emitting on x. The* fair testing preorder $\sqsubseteq_{\mathrm{fair}}$ *is the largest precongruence whose inverse refines the fair-must predicates $\Box{\Downarrow_x}$. The* fair testing equivalence $\simeq_{\mathrm{fair}}$ *is the largest congruence that refines the fair-must predicates $\Box{\Downarrow_x}$.*

$$P \,\square\!\Downarrow_x \stackrel{def}{=} \forall P' \,.\, P \to^* P' \textit{ implies } P' \Downarrow_x$$

$$P \sqsubseteq_{\text{fair}} Q \stackrel{def}{=} \forall C[\,], x.\, C[Q] \,\square\!\Downarrow_x \textit{ implies } C[P] \,\square\!\Downarrow_x$$

$$P \simeq_{\text{fair}} Q \stackrel{def}{=} \forall C[\,], x.\, C[Q] \,\square\!\Downarrow_x \textit{ if and only if } C[P] \,\square\!\Downarrow_x$$

For all processes P, if $P \,\square\!\Downarrow_x$ then $P \Downarrow_x$, and if there are no infinite computations, $P \,\square\!\downarrow_x$ and $P \,\square\!\Downarrow_x$ coincide. Fairness is hidden in the fair-must predicate: $P \,\square\!\Downarrow_x$ succeeds if there is still a way to emit on x after any reduction. Intuitively, the model is the set of barbs present on all finite and infinite fair traces. For instance, we have $\nu z.(\overline{z} \mid z.\overline{x} \mid !z.\overline{z}) \simeq_{\text{fair}} \overline{x}$, although the left-hand-side process has infinite reductions that never trigger $\overline{x}$.

By definition, may testing and fair testing equivalences are generally unrelated, but in the π-calculus fair testing is in fact strictly finer:

Proposition 11 *In the π-calculus, we have $\sqsubseteq_{\text{fair}} \subset \sqsubseteq_{\text{may}}$ and $\simeq_{\text{fair}} \subset \simeq_{\text{may}}$.*

PROOF. $\sqsubseteq_{\text{fair}} \subseteq \sqsubseteq_{\text{may}}$: we only have to prove that $\sqsubseteq_{\text{fair}}$ refines the barbs $\Downarrow_x$. We use the evaluation context $C[\,] \stackrel{\text{def}}{=} \nu r, z.(\overline{r}\langle y\rangle \mid x(\widetilde{v}).\overline{r}\langle z\rangle \mid r(u).\overline{u} \mid [\,])$ that transforms the presence of the barb $\Downarrow_x$ into the absence of the fair-must barb $\square\!\Downarrow_y$. For any processes P and Q of sort S with $r, z, y \notin S \cup \widetilde{v}$, we have $P \Downarrow_x$ if and only if $C[P] \not\square\!\Downarrow_y$. If $P \sqsubseteq_{\text{fair}} Q$ and $P \Downarrow_x$ then, by context-closure property, $C[P] \sqsubseteq_{\text{fair}} C[Q]$, $C[P] \not\square\!\Downarrow_y$, hence $C[Q] \not\square\!\Downarrow_y$ and $Q \Downarrow_x$.

The inclusions are strict, since $\overline{x} \oplus \mathtt{0} \simeq_{\text{may}} \overline{x}$ and $\overline{x} \oplus \mathtt{0} \not\sqsubseteq_{\text{fair}} \overline{x}$. □

Fair testing equivalence is also the largest congruence that refines both may- and fair-must- predicates. This property of fair testing also holds in CCS, in the join calculus, and for Actors, where a similar equivalence is proposed as the main semantics [5]. Conversely, we will establish that fair testing is strictly coarser than barbed congruence ($\approx \subset \simeq_{\text{fair}}$). Similar inclusions are established in [9,34]; the authors remark that weak bisimulation equivalences incorporate a particular notion of fairness, they identify sensitivity to the branching structure as an undesirable property of bisimulation, and they propose simulation-based sufficient conditions to establish fair testing.

In terms of discriminating power, fair testing is an appealing equivalence for asynchronous systems: it is stronger than may testing, detects deadlocks, but remains insensitive to termination and livelocks. (A process has a deadlock when it can reach a state with no observable behavior; in an asynchronous setting, this is independent of livelocks and termination.) In [10], for instance, distributed communication protocols are studied using the fair testing preorder as an implementation relation.

Note, however, that "abstract fairness" is not enforced by practical scheduling policies. Fair testing suffers from another technical drawback: direct proofs of equivalence are very difficult because they involve nested inductions for all quantifiers in the definition of fair-must tests in all evaluation contexts. The redeeming feature of fair testing is that it can be established using finer simulation-based equivalences. Precisely, we will establish a tight characterization of fair testing using *coupled simulations* in Section 6.

4.2 Coupled Simulations

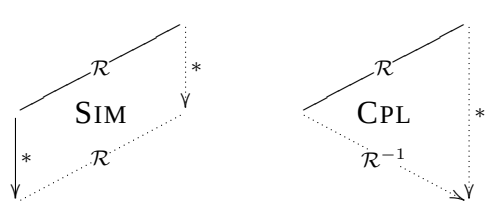
Independently of fair testing, labeled *coupled simulation* has been proposed in [37] to address similar issues; this simulation-based preorder does not require an exact correspondence between the internal choices, and thus abstracts some of the branching structure revealed by bisimulation. *Weakly-coupled simulation* is a variant that is insensitive to divergence [38]. It is used in [35] to establish the correctness of an encoding of the choice operator in the asynchronous π-calculus. (Here, we use barbed weakly-coupled simulations, and we consider a single, self-coupled simulation, rather than a pair of coupled simulations.)

Definition 12 *A relation $\mathcal{R}$ is a* barbed coupled simulation *when it is a barbed simulation that satisfies the coupling property: if $P\ \mathcal{R}\ Q$, then $Q \rightarrow^* \mathcal{R}\ P$.*

Barbed coupled similarity, *written $\dot{\lesssim}$, is the largest barbed coupled simulation.* Barbed coupled precongruence, *written $\lesssim$, is the largest precongruence that is a barbed coupled simulation.*

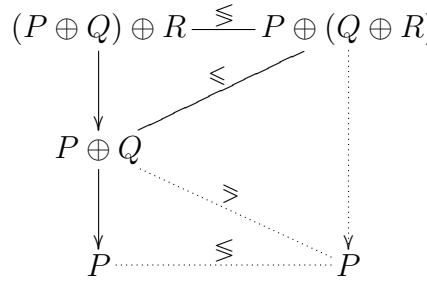
We write $\dot{\gtrsim} \stackrel{def}{=} \dot{\lesssim}^{-1}$, $\dot{\lesssim} \stackrel{def}{=} \dot{\lesssim} \cap \dot{\gtrsim}$, and $\gtrsim \stackrel{def}{=} \lesssim^{-1}$. Barbed coupled congruence, *written $\lesssim$, is $\lesssim \cap \gtrsim$.*

Using diagrams, the simulation and coupling requirements of the definition are:



If a coupled simulation $\mathcal{R}$ is also symmetric ($\mathcal{R} = \mathcal{R}^{-1}$), the coupling property is trivially verified, and $\mathcal{R}$ is in fact a bisimulation. Thus, for any reduction system, we have the inclusions $\dot{\approx} \subseteq \dot{\lesssim}$ and $\approx \subseteq \lesssim$.

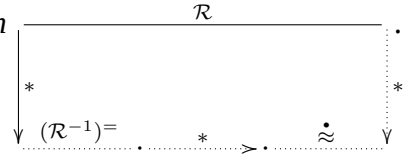
Typically, the discrepancy between $\leqslant$ and $\geqslant$ is used to describe processes that are in a transient state, bisimilar neither to the initial state nor to any final state. For example, for any processes P, Q, R (and up to $\approx$ after reducing $\oplus$):



In the π-calculus, we obtain that the inclusions are strict ($\dot{\approx} \subset \dot{\lesssim}$ and $\approx \subset \lesssim$) by comparing, for instance, $(\overline{x} \oplus \overline{y}) \oplus \overline{z}$ to $\overline{x} \oplus (\overline{y} \oplus \overline{z})$.

The "upward-reduction closure" relation $\leftarrow^*$ is always a barbed coupled simulation. We give a more general proof technique for establishing barbed coupled similarity using smaller candidate coupled simulations:

Lemma 13 (coupled simulation up to) *To establish $\mathcal{R} \subseteq \dot{\lesssim}$, it suffices to show that $\mathcal{R}$ refines the barbs $\Downarrow_x$ and satisfies the diagram*



PROOF. We show that $\phi \stackrel{\text{def}}{=} \dot{\approx} \leftarrow^* \mathcal{R}^= \dot{\approx}$ is a barbed coupled simulation:

- The relations $\dot{\approx}$, $\leftarrow$, and $\mathcal{R}$ all refine barbs, hence so does ϕ.
- By bisimulation on the left $\dot{\approx}$, we have $\leftarrow^* \phi \subseteq \phi$, so ϕ is a simulation.
- The diagram of the lemma trivially holds with $\mathcal{R}^=$ instead of $\mathcal{R}$ at the top. If $P \dot{\approx} (\leftarrow^* \mathcal{R}^=) \dot{\approx} Q$ then, using this diagram, $P \dot{\approx} (\mathcal{R}^{-1})^= \rightarrow^* \dot{\approx} \leftarrow^* \dot{\approx} Q$ and, by bisimulation on the right $\dot{\approx}$, we obtain Q' such that $P \dot{\approx} (\mathcal{R}^{-1})^= \rightarrow^* \dot{\approx} Q' \leftarrow^* Q$, that is, $P \ \phi^{-1} \ Q' \leftarrow^* Q$. □

As in the case of barbed bisimilarity in Section 3.2, there are two notions of congruence for barbed coupled similarity, but with a different outcome here:

Lemma 14 *In the π-calculus, we have $\lesssim \subset \dot{\lesssim}^\circ$.*

PROOF. The inclusion $\lesssim \subseteq \dot{\lesssim}^\circ$ holds by definition, as usual.

The discrepancy between the two congruences stems from internal choices that are spawned between visible actions. For instance, we prove that:

(1) $a.\overline{b} \oplus a.\overline{c} \ \dot{\lesssim}^\circ \ a.(\overline{b} \oplus \overline{c})$
(2) $a.\overline{b} \oplus a.\overline{c} \not\lesssim a.(\overline{b} \oplus \overline{c})$

Our proof illustrates the difficulty of dealing directly with $\dot{\leqslant}^{\circ}$, even for simple equations. We let AB and AC be the processes defined by reducing the first sum: $a.\overline{b} \oplus a.\overline{c} \to AB \approx a.\overline{b}$ and $a.\overline{b} \oplus a.\overline{c} \to AC \approx a.\overline{c}$. We begin with the second statement.

(2) Assume that we had $a.\overline{b} \oplus a.\overline{c} \leqslant a.(\overline{b} \oplus \overline{c})$. Then, the reduction $a.\overline{b} \oplus a.\overline{c} \to AB$ above must be simulated by no reduction (since $a.(\overline{b} \oplus \overline{c}) \not\to$). Moreover, $AB \not\to$, hence the two processes are coupled, and we have $a.\overline{b} \leqslant a.(\overline{b} \oplus \overline{c})$. By congruence property, for the evaluation context $\nu ab.(\overline{a} \,|\, [\,])$, we obtain

$$\nu ab.(\overline{a} \,|\, a.\overline{b}) \leqslant \nu ab.(\overline{a} \,|\, a.(\overline{b} \oplus \overline{c}))$$

and this equation is clearly false: only the process on the right has a barb $\Downarrow_c$ in two steps, hence these processes are not even may testing equivalent.

(1) After choosing a particular evaluation context $C[\,]$, however, the visible action yields a potential internal reduction. In our processes, interaction with the context is limited to reception on a; the context $C[\,]$ may interact with our processes if and only if there is another evaluation context $C'[\,]$ such that $C[\,] \to^* C'[\overline{a} \,|\, [\,]]$. We abbreviate this property as $C \Downarrow$.
We establish the equivalence above in a mostly co-inductive style by applying Lemma 13. We let $\mathcal{R}$ be the relation that contains the following pairs of processes: for every evaluation context $C[\,]$,

$$C[a.\overline{b} \oplus a.\overline{c}] \;\mathcal{R}\; C[a.(\overline{b} \oplus \overline{c})] \tag{3}$$

$$C[a.(\overline{b} \oplus \overline{c})] \;\mathcal{R}\; C[a.\overline{b} \oplus a.\overline{c}] \tag{4}$$

$$C[a.(\overline{b} \oplus \overline{c})] \;\mathcal{R}\; C[AB] \text{ when not } C \Downarrow \tag{5}$$

$$C[a.(\overline{b} \oplus \overline{c})] \;\mathcal{R}\; C[AC] \text{ when not } C \Downarrow \tag{6}$$

In (5) and (6), the condition on $C[\,]$ makes all related processes behave as $C[\mathtt{0}]$ (up to $\approx$). In particular, the requirements of Lemma 13 are easily met.

In (3,4), the requirement on barbs can be reformulated as the simple may testing equation $a.\overline{b} \oplus a.\overline{c} \simeq_{\text{may}} a.(\overline{b} \oplus \overline{c})$. The diagram requires more work. In (3), assume $C[a.\overline{b} \oplus a.\overline{c}] \to^* T$; We distinguish several cases:

(a) the sum $a.\overline{b} \oplus a.\overline{c}$ is not reduced. Hence the enclosing context cannot interact with this process and, for some other context $C'[\,]$, we have $C[\,] \to^* C'[\,]$ and $T \equiv C'[a.\overline{b} \oplus a.\overline{c}]$. The same series of reductions applied on the other side yields the process $C'[a.(\overline{b} \oplus \overline{c})]$. These two resulting processes are related by (4).

(b) the process $a.\overline{b} \oplus a.\overline{c}$ is reduced, e.g. $a.\overline{b} \oplus a.\overline{c} \to AB$, and
   (i) either the enclosing context does not interact with this AB, and instead we have $C[\,] \to^* C'[\,] \not\Downarrow$ and $T \equiv C'[AB]$. We perform the same series of reductions on the other side, and the two resulting processes are related by (5).
   (ii) or the context emits $\overline{a}$ that interacts with the resulting process AB. In that case, $C[a.(\overline{b} \oplus \overline{c})] \to^* \approx T$ by using the same series of re-

ductions, except for the internal choice which has to be deferred until communication on a enables it. The two resulting processes are bisimilar.

(iii) or this interaction does not occur, but the context can still emit on a: we have $T \equiv C'[AB] \rightarrow^* C''[\overline{a} \mid AB]$, thus $T \rightarrow^* \approx C''[\overline{b}]$, and we obtain reductions $C[a.\overline{b} \oplus a.\overline{c}] \rightarrow^* \approx C''[\overline{b}]$ as in the previous case.

In (4), we perform a similar case analysis, but the situation is simpler:

(a) The context does not interact with the process (which is inert in isolation); the two resulting processes are related by (3).

(b) The context provides a message $\overline{a}$ received by the process, and

(i) either the sum $\overline{b} \oplus \overline{c}$ is reduced in the following reductions. We anticipate the right choice in the left process by reducing to AB or AC, then apply the same series of reductions and obtain bisimilar processes.

(ii) or the internal choice is not reduced. We select any branch of the sum in the left process, then perform the same series of reductions. The two resulting processes are related by the reduction of $\overline{b} \oplus \overline{c}$ that chooses the same branch in the right process, up to bisimilarity. □

The exact relation between fair testing and barbed coupled congruence is intriguing. These equivalences are applied to the same problems, typically the study of distributed protocols where high-level atomic steps are implemented as a negotiation between distributed components, with several steps that perform a gradual commitment. Yet, their definitions are very different, and both have their advantages; fair testing is arguably more natural than coupled congruence, but lacks efficient proof techniques.

It is not too hard to establish that (the inverse of) barbed coupled simulations also refine fair-must barbs. The proof uses simulations in both directions, which somehow reflects the alternation of quantifiers in the definition of fair-must barbs.

Lemma 15 *In any reduction system, (1) the inverse of barbed coupled simulations refine all fair-must barbs: let $\mathcal{R}$ be a barbed coupled simulation. If $P\ \mathcal{R}^{-1}\ Q$ and $P\ \Box\Downarrow_x$, then also $Q\ \Box\Downarrow_x$. Hence, (2) the precongruence of barbed coupled similarity is finer than fair testing: $\dot{\leqslant}^\circ \subseteq \sqsubseteq_{\text{fair}}$.*

PROOF. (1) If $Q \rightarrow^* Q'$, these reductions can be simulated by $P \rightarrow^* P'\ \mathcal{R}^{-1}\ Q'$. Using the coupling condition, we also have $P' \rightarrow^* P''\ \mathcal{R}\ Q'$. By definition of $P\ \Box\Downarrow_x$, we have $P'' \Downarrow_x$. Finally, $\mathcal{R}$ refines weak barbs, and thus $Q' \Downarrow_x$.

(2) $\dot{\leqslant}$ refines all fair tests $\Box\Downarrow_x$, hence $\dot{\leqslant}^\circ$ refines them in any evaluation contexts. □

In the π-calculus, this precongruence is strictly finer than fair testing:

Lemma 16 *In the π-calculus, we have $\dot{\leqslant}^\circ \subset \sqsubseteq_{\text{fair}}$ and $\dot{\lesssim}^\circ \subset \simeq_{\text{fair}}$.*

PROOF. For instance, we have (1) $a \simeq_{\text{fair}} a \oplus \mathtt{0}$ but (2) $a \not\dot{\gtrsim}^\circ (a \oplus \mathtt{0})$:

(1) Since $a \oplus \mathtt{0} \rightarrow\!\!\approx a$, if $C[a \oplus \mathtt{0}]\ \Box\!\!\Downarrow_x$, then also $C[a]\ \Box\!\!\Downarrow_x$. Conversely, assume $C[a]\ \Box\!\!\Downarrow_x$. By induction on the number of reduction steps $n \geq 0$, we show that $C[a \oplus \mathtt{0}] \rightarrow^n Q$ implies $Q \Downarrow_x$ for all evaluation contexts C.
- $n = 0$: we have $Q = C[a \oplus \mathtt{0}] \rightarrow\!\!\approx C[a]$ and $C[a] \Downarrow_x$, hence $Q \Downarrow_x$.
- Inductive case: if $C[a \oplus \mathtt{0}] \rightarrow R \rightarrow^n Q$, then one of the following holds:
  (a) $R \equiv C'[a \oplus \mathtt{0}]$ and $C[a] \rightarrow C'[a]$ for some evaluation context C'. We have $C'[a]\ \Box\!\!\Downarrow_x$ and conclude by induction hypothesis.
  (b) $R \approx C[a]$, hence $Q \Downarrow_x$ by hypothesis $C[a]\ \Box\!\!\Downarrow_x$.
  (c) $R \approx C[\mathtt{0}]$, hence $Q \Downarrow_x$ by hypothesis $C[a]\ \Box\!\!\Downarrow_x$ and the general property that $C'[a] \Downarrow_x$ implies $C'[\mathtt{0}] \Downarrow_x$ (with the same reductions except the one that consumes the input a and a message $\overline{a}$).

(2) Otherwise, by applying the context $\overline{a} \,|\, [\ ]$, we would have $\overline{a} \,|\, a \dot{\gtrsim} \overline{a} \,|\, (a \oplus \mathtt{0})$. The step $\overline{a} \,|\, (a \oplus \mathtt{0}) \rightarrow\!\!\approx \overline{a}$ is simulated by some $\overline{a} \,|\, a \rightarrow^* T$ and, since $\overline{a} \Downarrow_a$, we have $T \equiv \overline{a} \,|\, a$. By coupling and simulation, we obtain the contradiction $\mathtt{0} \dot{\lesssim} \overline{a}$. □

Nonetheless, the distance between fair testing and barbed coupled precongruence is rather small. As we shall see in Section 6, both relations coincide in the join calculus, and can be made to coincide in the π-calculus with a small restriction on the barbs.

5 Equivalences with a Single Observation

We complete our exploration of asynchronous equivalences with a discussion of alternate definitions of observation. So far, we have used a specific output predicate $\downarrow_x$ for every name, but there are other natural choices. In the initial paper on barbed equivalences [32], and in most definitions of testing equivalences, a single predicate is used instead of an indexed family. Either there is a special observable action ω, or all barbs are merged into one. Accordingly, for every family of observation predicates (e.g., $\Downarrow_x$), we define an existential observation predicate that tests any of these predicates (e.g., $P \Downarrow \stackrel{\text{def}}{=} \exists x.P \Downarrow_x$) and, for every equivalence, we define its existential variant (e.g., $\dot{\approx}^\circ_\exists$) that refines only $\Downarrow$, at least by definition.

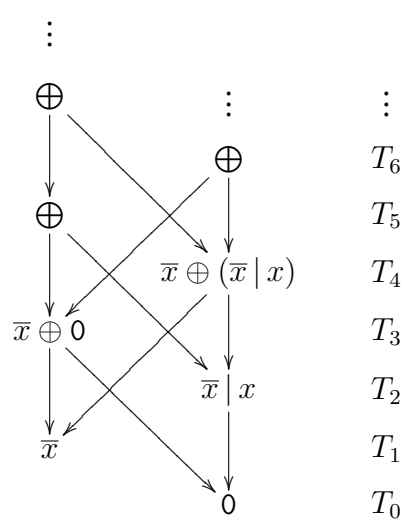
Existential equivalences that are closed by application of evaluation contexts usually coincide with their base equivalence. In the π-calculus, for example, for any process P of finite sort S, we have $\nu(S \setminus \{x\}).P \Downarrow$ if and only if $P \Downarrow_x$, and thus

we easily prove $\simeq_{\mathrm{may},\exists} = \simeq_{\mathrm{may}}$, $\simeq_{\mathrm{fair},\exists} = \simeq_{\mathrm{fair}}$, $\lesssim_\exists = \lesssim$, and $\approx_\exists = \approx$ using evaluation contexts $\nu\widetilde{x}.[\,]$. In contrast, when bisimulation and congruence are not jointly required in the definition, existential equivalences can be significantly coarser. The question arises for the existential variants of $\dot{\approx}^\circ$ and $\dot{\lesssim}^\circ$. Next, we establish the strict inclusion $\dot{\approx}^\circ_\exists \subset \approx_\exists$. Precisely, we show that weak $\exists$-barbed congruence is an *inductive*, or limit, bisimulation.

5.1 Equivalence Classes for $\dot{\approx}_\exists$

We first characterize the equivalence classes for existential barbed bisimilarity $\dot{\approx}_\exists$. In Section 7, we show that observing barbs on just two different names created a rich hierarchy of equivalence classes, from which an infinite set of prefix codes could be selected (Lemma 31). If only a single test is available, then this construction collapses.

In the π-calculus, besides the obviously-different processes $T_0 = \mathtt{0}$ and $T_1 = \overline{x}$, we have the process $T_2 = \overline{x} \,|\, x$ whose only and peculiar behavior is to rescind its $\downarrow_x$-barb to become $\mathtt{0}$. Starting from these three processes, one can construct a quasi-linear sequence of processes, setting $T_{i+3} = T_i \oplus T_{i+1}$.



In addition, we can code a limit process $T_\omega \approx \bigoplus_{i\in\mathbb{N}} T_i$, as detailed below—by induction on n, the equivalence given after the definition holds for any $n \geq 0$.

$$\begin{aligned} T_\omega \stackrel{\text{def}}{=}\ & \nu s, t_0, t_1, t_2.(s(t).\bar{t} \,|\, \textstyle\prod_{i=0}^{2}(\overline{s}\langle t_i\rangle \,|\, t_i.T_i) \,| \\ & \nu g.(\overline{g}\langle t_0, t_1, t_2\rangle \,|\, !g(a,b,c).\nu d.(\overline{g}\langle b,c,d\rangle \,|\, \overline{s}\langle d\rangle \,|\, d.(\overline{a} \oplus \overline{b})))) \end{aligned}$$

$$\begin{aligned} T_\omega \approx\ & \nu s, t_0, \ldots, t_{n+2}.(s(t).\bar{t} \,|\, \textstyle\prod_{i=0}^{n+2}(\overline{s}\langle t_i\rangle \,|\, t_i.T_i) \,| \\ & \nu g.(\overline{g}\langle t_n, t_{n+1}, t_{n+2}\rangle \,|\, !g(a,b,c).\nu d.(\overline{g}\langle b,c,d\rangle \,|\, \overline{s}\langle d\rangle \,|\, d.(\overline{a} \oplus \overline{b})))) \end{aligned}$$

We use these processes to partition processes, as follows:

Definition 17 *The* signature $T(P)$ *of a process* P *is the set of indices of the* T_i *reachable from* P *up to bisimilarity:*

$$T(P) = \{i \in \mathbb{N} \mid \ P \rightarrow^* \dot{\approx}_\exists T_i \ \}$$

By definition, $\dot{\approx}_\exists$-equivalent processes must have the same signature; the converse also holds, because the sequence $(T_i)_{i \leq \omega}$ spans exactly the equivalence classes of $\dot{\approx}_\exists$:

Proposition 18 *For any* π*-calculus process* P*, there is a unique* $j \in \mathbb{N} \cup \{\omega\}$ *such that* $P \dot{\approx}_\exists T_j$ *and, moreover, if* $j = \omega$ *then* $T(P) = \mathbb{N}$*; if* $j < \omega$ *then* $j = \max T(P)$ *and* $T(P) = \{0, \ldots, j-2, j\}$.

PROOF. First, note that by construction of the $(T_n)_{n \in \mathbb{N}}$, we have $T_n \rightarrow^* T'$ if and only if $T' \approx T_i$ for some $i \in \{0, \ldots, n-2, n\}$; also, $T_\omega \rightarrow^* T'$ if and only $T' \approx T_i$ for some $i \leq \omega$. The second half of the proposition therefore follows directly from the first half.

Let P be any process; if $P \not\Downarrow$ then $P \approx T_0$. Otherwise, we show that if $T(P) \neq \mathbb{N}$, then $P \dot{\approx}_\exists T_{n+1}$, where $n = \min(\mathbb{N} \setminus T(P))$, by induction on n. Assuming this holds for all $i < n$, we define the relation $\mathcal{R}^n = \{(P, T_{n+1}) | P \Downarrow \text{ and } n = \min(\mathbb{N} \setminus T(P))\}$ and show that $\mathcal{R}^n \cup \dot{\approx}_\exists$ is a barbed existential bisimulation. Suppose $P \ \mathcal{R}^n \ T_{n+1}$; then both $P \Downarrow$ and $T_{n+1} \Downarrow$. If $P \rightarrow^* P'$, then either $P' \ \mathcal{R}^n \ T_{n+1}$, or $P' \dot{\approx}_\exists T_i$ for some $i \leq n$: take $i = 0$ if $P' \not\Downarrow$ and $i = 1 + \min(\mathbb{N} \setminus T(P'))$ otherwise. Furthermore $i \in T(P)$, so actually $i < n$, and then $T_{n+1} \rightarrow \approx T_i$. Conversely, if $T_{n+1} \rightarrow^* T'$ then $T' \approx T_i$ for some $i < n$, hence $i \in T(P)$, that is, $P \rightarrow^* \dot{\approx}_\exists T_i$. Similarly, the relation $\{(P, T_\omega) \mid T(P) = \mathbb{N}\} \cup \dot{\approx}_\exists$ is a barbed existential bisimulation.

This shows the existence of a j such that $P \dot{\approx}_\exists T_j$; this j is unique, because T_0 is the only $T_i \not\Downarrow$, T_ω is the only T_i such that $T(T_i) = \mathbb{N}$, and T_{n+1} is the only T_i such that $n + 1 \in T(T_i)$ but $n \notin T(T_i)$. □

Next, we provide a shortcut for computing $\max T(P)$ for a set of processes $P \in \mathcal{A}$ without actually performing a bisimulation proof. Intuitively, the sets $\mathcal{B}_i$ contain processes that *must* be reached from $\mathcal{A}$, whereas the $\mathcal{B}'_i$ contain additional processes that *may* be reached from $\mathcal{A}$. Moreover, all processes in any given set $\mathcal{B}_i$, $\mathcal{B}'_i$ are equivalent.

Lemma 19 *For* $k > 0$ *and* $k' \geq 0$*, let* $\mathcal{A}, \mathcal{B}_1, \ldots, \mathcal{B}_k, \mathcal{B}'_1, \ldots, \mathcal{B}'_{k'}$ *be sets of processes such that*

(a) *for any $P \in \mathcal{A}$, and any i, $1 \leq i \leq k$, we have $P \rightarrow^* Q$ for some $Q \in \mathcal{B}_i$;*
(b) *for any $P \in \mathcal{A}$, if $P \rightarrow Q$, then $Q \in \mathcal{A} \cup \mathcal{B}_1 \cup \ldots \cup \mathcal{B}_k \cup \mathcal{B}'_1 \cup \ldots \cup \mathcal{B}'_{k'}$;*
(c) *there are integers $j_1, \ldots, j_k$, $j'_1, \ldots, j'_{k'}$ such that*
$\max T(Q) = j_i$ *for all $Q \in \mathcal{B}_i$, and*
$\max T(Q) = j'_i$ *for all $Q \in \mathcal{B}'_i$.*

Let $J = \{j_1, \ldots, j_k\}$, $J' = \{j'_1, \ldots, j'_{k'}\}$, $j = \max J \cup J'$, and assume $j > 0$. We have:

(1) *if $j - 1 \notin J \cup J'$ and $j \in J$ then $\max T(P) = j$ for all $P \in \mathcal{A}$;*
(2) *if $j - 1 \notin J \cup J'$ and both $j - 2, j - 3 \in J$ then $\max T(P) = j$ for all $P \in \mathcal{A}$;*
(3) *if both $j, j - 1 \in J$ then $\max T(P) = j + 2$ for all $P \in \mathcal{A}$.*

PROOF. Let j' stand for j in cases (1) and (2), and $j + 2$ in case (3). We will successively establish that $j' \in T(P)$ for all $P \in \mathcal{A}$, then that $j' - 1 \notin T(P)$ for all $P \in \mathcal{A}$. Then for any $P \in \mathcal{A}$, the only possibility allowed by Proposition 18 for $T(P)$ is $\{0, \ldots, j' - 2, j'\}$, whence $\max T(P) = j'$.

To show that $j' \in T(P)$ for any $P \in \mathcal{A}$, we first note that conditions (a) and (c) imply $T(P) \supseteq J$. Thus in case (1) we have $j' = j \in T(P)$; in cases (2) and (3) we only get $\{j' - 2, j' - 3\} \subseteq T(P)$, but then Proposition 18 implies that we must also have $j' \in T(P)$.

Now assume that $j' - 1 \in T(P)$ for some $P \in \mathcal{A}$; by definition there must be some $P' \stackrel{\cdot}{\approx}_\exists T_{j'-1}$ such that $P \rightarrow^* P'$. Now $T(P') = T(T_{j'-1}) \not\ni j'$, so $P' \notin \mathcal{A}$ by the first part of the proof. Let Q be the first process in the reduction $P \rightarrow^* P'$ that is not in $\mathcal{A}$, then by condition (b) Q is in some $\mathcal{B}_i$ or $\mathcal{B}'_i$; in either case, by condition (c) we have $\max T(Q) = j''$ for some $j'' \in J \cup J'$, and $T(Q) = \{0, \ldots, j'' - 2, j''\}$ by Proposition 18. But $j' - 1 \in T(Q)$ since $Q \rightarrow^* P'$, and since $j' \geq j \geq j''$ we must have $j' - 1 = j''$. In cases (1) and (2) this would mean $j'' = j - 1$, which is ruled out, and in case (3), $j'' = j + 1$, which is also impossible. □

5.2 Limit Characterization

Definition 20 *Inductive bisimilarity is the limit of the monotone operator associated with the definition of barbed bisimulation:*

$$
\begin{aligned}
P \stackrel{\cdot}{\approx}^0 Q &\stackrel{def}{=} \forall x \,.\, P \Downarrow_x \textit{ iff } Q \Downarrow_x \\
P \stackrel{\cdot}{\approx}^{n+1} Q &\stackrel{def}{=} P \rightarrow^* P' \textit{ implies } Q \rightarrow^* \stackrel{\cdot}{\approx}^n P' \textit{ and, conversely,} \\
&\qquad Q \rightarrow^* Q' \textit{ implies } P \rightarrow^* \stackrel{\cdot}{\approx}^n Q' \\
P \stackrel{\cdot}{\approx}^\omega Q &\stackrel{def}{=} \forall n \,.\, P \stackrel{\cdot}{\approx}^n Q
\end{aligned}
$$

By definition, this limit bisimilarity is coarser than the co-inductive one: $\dot{\approx} \subseteq \dot{\approx}^{\omega}$. In the π-calculus, as usual, this inclusion is strict, although the two equivalences coincide for all image-finite processes. Consider, for instance, $P_j \stackrel{\text{def}}{=} T_j \oplus \overline{y}$, $P \approx \bigoplus_{i\in\mathbb{N}} P_i$, and $Q \approx \bigoplus_{i\in\mathbb{N}\cup\{\omega\}} P_i$ (where $y \neq x$, and the infinite sum can be coded as above). The reduction $Q \rightarrow \dot{\approx}\ T_\omega$ cannot be simulated by P, hence $P \not\dot{\approx}\ Q$. Conversely, for any $j > n$, we have $T_j \dot{\approx}^n T_\omega$, and thus $P \dot{\approx}^{\omega} Q$.

There are several other ways to define limit bisimilarity and its congruence. For instance, one can define a "reduction-based" limit equivalence with a context-closure property, $\approx^{\omega}$, such that, at every level, an evaluation context can be applied before bisimulation. In fact, our limit bisimilarity is very weak. A variant of the above example shows that $(\dot{\approx}^{\omega})^{\circ}$ is strictly weaker than $\approx^{\omega}$, and thus weaker than the classical (labeled) limit bisimilarity, as defined for CCS by Milner [28].

Theorem 2 *In the π-calculus, we have $\dot{\approx}^{\circ}_{\exists} = (\dot{\approx}^{\omega})^{\circ}$.*

The proof that $\dot{\approx}^{\circ}_{\exists} \supseteq (\dot{\approx}^{\omega})^{\circ}$ is fairly easy, since by induction on n, $P \dot{\approx}^n Q \dot{\approx}_{\exists} T_n$ implies $P \dot{\approx}_{\exists} T_n$. For the converse, we need to show that for any processes P, Q, if $C[P] \dot{\approx}_{\exists} C[Q]$ for all execution contexts $C[\ ]$, then $P \dot{\approx}^{\omega} Q$. Clearly it is enough to show this for all P, Q whose free names lie in a fixed, finite, but arbitrary set S. Let $\mathcal{U}$ be the set of these processes. Having fixed S, we can use contexts of the form $\nu\widetilde{S}.(C|[\ ])$; in the following, we write $C\|P$ for the process $\nu\widetilde{S}.(C|P)$. We also assume without loss of generality that the channel x used in the construction of the T_n is not in S.

We thus need to find processes C that will allow us to refine the equivalence classes of $\dot{\approx}_{\exists}$ into those of $\dot{\approx}^{\omega}$. We use the definition below:

Definition 21 *A subset $\mathcal{A}$ of $\mathcal{U}$ is* separable *at some $N \in \mathbb{N}$, using a sequence $(C_m)_{m\geq N+2}$ of processes, when, for any $P \in \mathcal{U}$ and any integer $m \geq N+2$, we have $\max T(C_m\|P) = m$ if $P \in \mathcal{A}$ and $\max T(C^{\mathcal{A}}_m\|P) = N$ if $P \notin \mathcal{A}$.*

For example, we have:

(1) The set $\mathcal{U}$ itself is separable at $N_{\mathcal{U}} = 0$, using $C^{\mathcal{U}}_m = T_m$.
(2) For any $y \in S$, the set $\mathcal{A}_y = \{P \in \mathcal{U} \mid P \Downarrow_y\}$ is separable at $N_{\mathcal{A}_y} = 0$, using $C^{\mathcal{A}_y}_m = y(\widetilde{z}).T_m$.
(3) Any separable set $\mathcal{A}$ is closed by inverse reduction: if $\mathcal{A}$ is separable at N using $(C_m)_{m\geq N+2}$ and $P \rightarrow Q \in \mathcal{A}$, then $N+2 = \max T(C_{N+2}\|Q) \leq \max T(C_{N+2}\|P)$, so $\max T(C_{N+2}\|P) \neq N$, hence $P \in \mathcal{A}$.

Furthermore, the separation index N can always be increased:

Lemma 22 *If $\mathcal{A}$ is separable at some N, using $(C_m)_{m\geq N+2}$, it is also separable at any $N' \geq N+2$, using $(C'_m)_{m\geq N'+2} = (C_m \oplus T_{N'})_{m\geq N'+2}$.*

PROOF. We use Lemma 19 to compute the signatures. For any $m \geq N' + 2$, consider $\mathcal{A}' = \{C'_m \| P \mid P \notin \mathcal{A}\}$; any $C'_m \| P \in \mathcal{A}'$ must must have reductions to both $T_{N'} \| P$ and $C_m \| P$; all other derivatives remain in $\mathcal{A}'$, since $\mathcal{A}$ is closed by inverse reduction. By assumption $\max T(C_m \| P) = N$ for any $P \notin \mathcal{A}$, and $\max T(T_{N'} \| P) = N' \geq N + 2$, so by Lemma 19, case (1), we have $\max T(C_m \| P) = N'$.

Now consider $\mathcal{A}'' = \{C'_m \| P \mid P \in \mathcal{A}\}$; any $C_m \| P \in \mathcal{A}''$ must must have reductions to both $T_{N'} \| P$ and $C_m \| P$; it may also reduce to some $C_m \| Q \in \mathcal{A}'$. By assumption $\max T(C_m \| P) = m \geq N' + 2$ for any $P \in \mathcal{A}$, and $\max T(T_{N'} \| P) = N'$, and we have just shown that $\max T(C_m \| Q) = N'$, so again by Lemma 19, case (1), we have $\max T(C_m \| P) = m$. □

Let us denote by $[\mathcal{A}]_{\Leftarrow}$ the closure under inverse reduction of $\mathcal{A} \subseteq \mathcal{U}$, i.e., $[\mathcal{A}]_{\Leftarrow} = \{P \mid \exists Q \in \mathcal{A}, P \to^* Q\}$. Up to this closure, separable sets are closed under set theoretical operations:

Proposition 23 *If $\mathcal{A}$ and $\mathcal{B}$ are separable, then $\mathcal{A} \cup \mathcal{B}$, $\mathcal{A} \cap \mathcal{B}$, and $[\mathcal{A} \setminus \mathcal{B}]_{\Leftarrow}$ are separable.*

PROOF. Using Lemma 22, we can assume that $\mathcal{A}$ and $\mathcal{B}$ are separable at the *same* $N \geq 2$, using $(C^{\mathcal{A}}_m)_m$ and $(C^{\mathcal{B}}_m)_m$, respectively.

This immediately implies that $\mathcal{A} \cup \mathcal{B}$ is separable at N, using $(C_m)_m = (C^{\mathcal{A}}_m \oplus C^{\mathcal{B}}_m)_m$: as in Lemma 22, we directly apply Lemma 19 to compute the signatures.

For the intersection, the contexts $(C_m)_m$ need to be defined recursively, using the formula $C_{m+3} = C_m \oplus C_{m+1}$, except for the base cases appearing in the first column of the table below.

The four other columns of the table give the value of $\max T(C_m \| P)$, for P in $\mathcal{U} \setminus (\mathcal{A} \cup \mathcal{B})$, $\mathcal{A} \setminus \mathcal{B}$, $\mathcal{B} \setminus \mathcal{A}$, and $\mathcal{A} \cap \mathcal{B}$, respectively. The table can be filled in top-down, left-to-right, using Lemma 19. The lines for the base cases C_{N+2}, C_{N+3}, C_{N+4}, and C_{N+6} can be filled in directly from the assumptions. For the other lines, we note that $C_m \| P$ must reduce to both $C_{m-2} \| P$ and $C_{m-3} \| P$, and may reduce to some $C_m \| Q$, if $P \to Q$, which is in a different cell only if $Q \in \mathcal{U} \setminus (\mathcal{A} \cup \mathcal{B})$ or $P \in \mathcal{A} \cap \mathcal{B}$. Extending, by induction on m, the pattern established in the last three lines of the table completes the proof that $\mathcal{A} \cap \mathcal{B}$ is separable at $N + 6$ using $(C_m)_{m \geq N+8}$.

The computation for $[\mathcal{A} \setminus \mathcal{B}]_{\Leftarrow}$ is similarly summarized in the table below. Note that the $\mathcal{A} \cap \mathcal{B}$ column has been partitioned into $\mathcal{A} \setminus [\mathcal{A} \setminus \mathcal{B}]_{\Leftarrow}$ and $\mathcal{B} \cap [\mathcal{A} \setminus \mathcal{B}]_{\Leftarrow}$ columns: obviously $\mathcal{A} \setminus [\mathcal{A} \setminus \mathcal{B}]_{\Leftarrow} \subseteq \mathcal{B}$, and $[\mathcal{A} \setminus \mathcal{B}]_{\Leftarrow} \subseteq \mathcal{A}$ because of the closure property of $\mathcal{A}$. The latter inclusion also implies $[\mathcal{A} \setminus \mathcal{B}]_{\Leftarrow} = (\mathcal{A} \setminus \mathcal{B}) \cup (\mathcal{B} \cap [\mathcal{A} \setminus \mathcal{B}]_{\Leftarrow})$. The

separating sequence	$\mathcal{U} \setminus (\mathcal{A} \cup \mathcal{B})$	$\mathcal{A} \setminus \mathcal{B}$	$\mathcal{B} \setminus \mathcal{A}$	$\mathcal{A} \cap \mathcal{B}$
$C_{N+2} \equiv C^{\mathcal{A}}_{N+2}$	N	$N+2$	N	$N+2$
$C_{N+3} \equiv C^{\mathcal{B}}_{N+3}$	N	N	$N+3$	$N+3$
$C_{N+4} \equiv C^{\mathcal{A}}_{N+4}$	N	$N+4$	N	$N+4$
$C_{N+5} \equiv C_{N+2} \oplus C_{N+3}$	N	$N+2$	$N+3$	$N+5$
$C_{N+6} \equiv T_{N+6}$	$N+6$	$N+6$	$N+6$	$N+6$
$C_{N+7} \equiv C_{N+4} \oplus C_{N+5}$	N	$N+4$	$N+3$	$N+7$
$C_{N+8} \equiv C_{N+5} \oplus C_{N+6}$	$N+6$	$N+6$	$N+6$	$N+8$
$C_{N+9} \equiv C_{N+6} \oplus C_{N+7}$	$N+6$	$N+6$	$N+6$	$N+9$
$C_{N+10} \equiv C_{N+7} \oplus C_{N+8}$	$N+6$	$N+6$	$N+6$	$N+10$

separating sequence	$\mathcal{U} \setminus (\mathcal{A} \cup \mathcal{B})$	$\mathcal{A} \setminus \mathcal{B}$	$\mathcal{B} \setminus \mathcal{A}$	$\mathcal{A} \setminus [\mathcal{A} \setminus \mathcal{B}]_{\Leftarrow}$	$\mathcal{B} \cap [\mathcal{A} \setminus \mathcal{B}]_{\Leftarrow}$
$C_N \equiv C^{\mathcal{B}}_{N+6}$	N	N	$N+6$	$N+6$	$N+6$
$C_{N+1} \equiv C^{\mathcal{A}}_{N+3}$	N	$N+3$	N	$N+3$	$N+3$
$C_{N+2} \equiv C^{\mathcal{A}}_{N+2}$	N	$N+2$	N	$N+2$	$N+2$
$C_{N+3} \equiv C_N \oplus C_{N+1}$	N	$N+3$	$N+6$	$N+6$	$N+6$
$C_{N+4} \equiv C^{\mathcal{A}}_{N+4}$	N	$N+4$	N	$N+4$	$N+4$
$C_{N+5} \equiv C_{N+2} \oplus C_{N+3}$	N	$N+5$	$N+6$	$N+6$	$N+8$
$C_{N+6} \equiv T_{N+6}$	$N+6$	$N+6$	$N+6$	$N+6$	$N+6$
$C_{N+7} \equiv C_{N+4} \oplus C_{N+5}$	N	$N+7$	$N+6$	$N+6$	$N+10$
$C_{N+8} \equiv C_{N+5} \oplus C_{N+6}$	$N+6$	$N+8$	$N+6$	$N+6$	$N+8$
$C_{N+9} \equiv T_{N+9}$	$N+9$	$N+9$	$N+9$	$N+9$	$N+9$
$C_{N+10} \equiv C_{N+7} \oplus C_{N+8}$	$N+6$	$N+10$	$N+6$	$N+6$	$N+10$
$C_{N+11} \equiv C_{N+8} \oplus C_{N+9}$	$N+9$	$N+11$	$N+9$	$N+9$	$N+11$
$C_{N+12} \equiv C_{N+9} \oplus C_{N+10}$	$N+9$	$N+12$	$N+9$	$N+9$	$N+12$
$C_{N+13} \equiv C_{N+10} \oplus C_{N+11}$	$N+9$	$N+13$	$N+9$	$N+9$	$N+13$

signature computations are similar to those for the $\mathcal{A} \cap \mathcal{B}$ table; however they rely on the fact that if $P \in \mathcal{B} \cap [\mathcal{A} \setminus \mathcal{B}]_{\Leftarrow}$, then $C_m \| P$ *must* reduce to some $C_m \| Q$ with $Q \in \mathcal{A} \setminus \mathcal{B}$, while if $P \in \mathcal{A} \setminus [\mathcal{A} \setminus \mathcal{B}]_{\Leftarrow}$ there *cannot* be a transition $P \to Q \in \mathcal{A} \setminus \mathcal{B}$. Again, the table is extended by induction to establish that $[\mathcal{A} \setminus \mathcal{B}]_{\Leftarrow}$ is separable at $N+9$ using $(C_m)_{m \geq N+11}$.

Proof of Theorem 2 For any $Q \in \mathcal{U}$, we show by induction on n that the set $I_n(Q) = \{P \in \mathcal{U} \mid P \rightarrow^* \dot{\approx}^n Q\}$ is separable, and that there is a finite number of such sets (that is, $\{I_n(Q) | Q \in \mathcal{U}\}$ is finite). For $n = 0$, we just have

$$I_0(Q) = \left[\left(\mathcal{U} \cap \bigcap_{Q\Downarrow_y} \mathcal{A}_y\right) \setminus \left(\bigcup_{Q\not\Downarrow_z} \mathcal{A}_z\right)\right]_{\Leftarrow}$$

which is separable by the above, and there are $2^{|S|}$ such sets. For the inductive case, we have

$$I_{n+1}(Q) = \left[\left(\bigcap_{Q\rightarrow^* Q'} I_n(Q')\right) \setminus \left(\bigcup_{Q\notin I_n(R)} I_n(R)\right)\right]_{\Leftarrow}$$

and there are at most $3^{|\{I_n(Q)|Q\in\mathcal{U}\}|}$ such sets.

We conclude by contradiction: assume $P \dot{\approx}^{\circ}_{\exists} Q$ and not $P\ (\dot{\approx}^{\omega})^{\circ}\ Q$. For some $P' = C[P]$, $Q' = C[Q]$, and $n \geq 0$, we have $P' \dot{\not\approx}^{n+1} Q'$ and $P' \dot{\approx}^{\circ}_{\exists} Q'$. (Since $\dot{\approx}^{\circ}_{\exists} \subseteq \dot{\approx}^0$, we cannot have $P' \dot{\not\approx}^0 Q'$.)

By definition of $\dot{\approx}^{n+1}$, there exists some process R such that $Q' \rightarrow^* R$ and not $P' \rightarrow^* \dot{\approx}^n R$, hence $Q' \in I_n(R)$ and $P' \notin I_n(R)$ (or conversely some process R such that $P' \in I_n(R)$ and $Q' \notin I_n(R)$). The set $I_n(R)$ is separable at some N using some $(C_m)_{m\geq N+2}$, hence for any $m \geq N + 2$ we have $C_m\|P' \dot{\approx}_{\exists} T_N$ and $C_m\|Q' \dot{\approx}_{\exists} T_m$.

Using $P' \dot{\approx}^{\circ}_{\exists} Q'$ with the context $\nu S.(C_m|[\ ])$ finally yields $T_N \dot{\approx}_{\exists} T_m$, which contradicts Proposition 18. □

6 Committed Barbs

Another variation on barbs is directly inspired by the join calculus. Assuming that the basic observation predicates reveal the outcome of a computation, rather than its transient state, one may be interested only in *committed observations* that cannot be rescinded by the process being observed. As regards asynchronous equivalences, this can be abstractly achieved by adding a requirement to the definition of barbs:

Definition 24 *A strong barb $\downarrow_x$ is* committed *when $P \downarrow_x$ implies $P \,\Box\Downarrow_x$.*

By extension, we say that a barb $\Downarrow_x$ is *committed* when its defining strong barb is committed. In the join calculus, the locality property guarantees that messages sent on free names are never used in internal reductions. Hence, strong barbs are refined by the reduction relation ($P \downarrow_x$ and $P \rightarrow^* P'$ implies $P' \downarrow_x$), and barbs are always committed.

In the π-calculus, the situation is not so simple. Consider the process $T_2 = \overline{x}\,|\,x$ used in Section 5: we have $P \downarrow_x$ and $T_2 \rightarrow \mathtt{0}$, hence clearly not $P \rightarrow^* \Box\Downarrow_x$. This

phenomenon is unfortunate, since names in the π-calculus are typically either intended for internal steps or for interaction with the environment, but not both. In the rest of this section, we therefore only test and compare processes that comply with a locality restriction that excludes communication on free names.

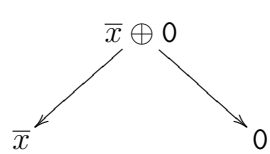
Definition 25 *A π-calculus process is* local *when reception occurs only on names bound by a restriction (not on free names and not on received names).*

The local π-calculus is the subcalculus of local processes. It is closed by structural equivalence, reduction, and application of local contexts. All barbs in the local π-calculus are committed.

The local π-calculus is not as limited as it may seem. In fact, most of the encodings appearing in this paper are expressed using only local terms. Except for Sections 5.1 and 5.2, reconsidered below, and Section 8, which requires a labeled semantics, all our definitions, results, and proofs apply unchanged to local π-calculus processes. Besides, one can design labeled semantics that are compatible with locality, then obtain results similar to those of Section 8 [27,17].

6.1 Bisimilarity and Fair Testing

We first reconsider the existential bisimilarity, $\dot{\approx}_\exists$, with a single committed barb. The resulting equivalence is far less exotic than with transient barbs; it has only three classes. The situation is displayed below for the local π-calculus:



Theorem 3 *In any reduction system, the barbed bisimilarity $\dot{\approx}_\exists$ that refines a single, committed barb $\Downarrow$ partitions processes into at most three classes characterized by $\Box\Downarrow$, $\not\Downarrow$, and $\Downarrow \wedge \not\Box\Downarrow$. In the local π-calculus, we have $\dot{\approx}^\circ_\exists = \simeq_{\rm fair}$.*

PROOF. The three predicates of the lemma induce a partition on processes; let $\mathcal{R}$ be the corresponding equivalence relation. We check that $\mathcal{R} \subseteq \dot{\approx}_\exists$ by establishing that $\mathcal{R}$ is a single-committed-barbed bisimulation.

- $\mathcal{R}$ refines the barb $\Downarrow$ by construction: it refines $\{\Downarrow,\not\Downarrow\}$ by splitting the first class in two, according to the predicate $\Box\Downarrow$, which always implies $\Downarrow$.
- $\mathcal{R}$ is a weak bisimulation: the two lower classes $\Box\Downarrow$ and $\not\Downarrow$ are closed by reduction, hence processes in these classes are trivially bisimilar. Besides, processes in the upper class always have reductions leading to both lower classes: $P \Downarrow$

implies $P \rightarrow^* P' \,\Box\!\!\Downarrow$ follows from the definition of committed barbs, and $P \not\Box\!\!\Downarrow$ is $P \rightarrow^* P' \not\Downarrow$.

In the local π-calculus, $0 \;\dot{\not\approx}_\exists\; \overline{x}$ and all three classes are separated by $\dot{\approx}_\exists$, hence $\dot{\approx}_\exists = \mathcal{R}$. Fair testing equivalence refines $\Box\!\!\Downarrow$ by definition and $\Downarrow$ by Lemma 11, hence $\simeq_{\text{fair}} \subseteq \dot{\approx}^\circ_\exists$. Conversely, the number of barbs makes no difference for $\simeq_{\text{fair}}$, which is a congruence, hence $\simeq_{\text{fair}} = \simeq_{\text{fair},\exists} \supseteq \dot{\approx}^\circ_\exists$. □

6.2 The Semantics of Coupled Simulation

Next, we provide another, more useful characterization of the $\simeq_{\text{fair}}$ equivalence. We establish that, in the local π-calculus, we have $\sqsubseteq_{\text{fair}} = \dot{\leqslant}^\circ$. To prove $\sqsubseteq_{\text{fair}} \subseteq \dot{\leqslant}^\circ$, we develop a semantic model of coupled similarity with committed barbs. We first consider processes whose behavior is especially simple. We say that a process P is *committed* when, for all tests $\Downarrow_x$, we have $P \Downarrow_x$ if and only if $P \,\Box\!\!\Downarrow_x$. Then, no reduction may visibly affect P: let S be the set of names

$$S \stackrel{\text{def}}{=} \{x \mid P \Downarrow_x\} = \{x \mid P \,\Box\!\!\Downarrow_x\}$$

For all P', if $P \rightarrow^* P'$, then P' is still committed to S. In a sense, P has converged to S, which entirely captures its behavior.

To every process P, we now associate the semantics $P^\flat$, defined as the set of sets of names S, for all committed derivatives of P:

$$P^\flat \stackrel{\text{def}}{=} \{S \subseteq \mathcal{N} \mid \exists P'.\, P \rightarrow^* P' \text{ and } S = \{x \mid P' \Downarrow_x\} = \{x \mid P' \,\Box\!\!\Downarrow_x\}\}$$

For example, $0^\flat$ is the singleton $\{\emptyset\}$ and $(\overline{x} \oplus \overline{y})^\flat$ is $\{\{x\}, \{y\}\}$. As is the case for weak barbs, $P^\flat$ decreases with reduction.

Remark 26 *In the π-calculus, $P^\flat \neq \emptyset$.*

PROOF. Although we use the remark only for local processes, our proof applies to any pi calculus process P. Consider all series of processes P_i for $i = 0 \ldots n$ ($n \geq 0$) such that

$$P = P_0 \rightarrow^* P_1 \rightarrow^* \cdots \rightarrow^* P_i \rightarrow^* \cdots \rightarrow^* P_n = P'$$

and such that the size of $\{x \mid P_i \Downarrow_x\}$ strictly decreases with i. We have at least one such series, P_0 for $n = 0$, and n is bounded by the number of free names in P, so there exists a series of maximal length. By construction, the process P' is then committed and yields an element of $P^\flat$. □

By definition, our semantics is closely related to the testing preorders:

Lemma 27 *Let $\subseteq_\flat$ be the preorder defined as $P \subseteq_\flat Q \stackrel{def}{=} P^\flat \subseteq Q^\flat$. In the local π-calculus, we have $\subseteq_\flat^\circ = \sqsubseteq_{\text{fair}}$.*

PROOF. The predicates $\Downarrow_x$ and $\Box\Downarrow_x$ can be recovered as follows: we have $P \Downarrow_x$ if and only if $x \in \bigcup P^\flat$, and $P \Box\Downarrow_x$ if and only if $x \in \bigcap P^\flat$. By definition of may testing and fair testing preorders, we thus obtain $\subseteq_\flat^\circ \subseteq \sqsubseteq_{\text{fair}} \subseteq \sqsubseteq_{\text{may}}$.

In the local π-calculus, the first inclusion is an equality. For any finite sets of names S and N such that $S \subseteq N$ and $t, t' \notin N$, we use the evaluation context:

$$T_S^N[\ ] \stackrel{\text{def}}{=} \nu S, t.\left([\ ] \mid \prod_{x \in S} (\overline{t} \mid x.t) \mid \prod_{x \in N \setminus S} x.(\overline{x} \mid \overline{t}) \mid t.\overline{t}' \right)$$

We check that $T_S^N[\ ]$ fair-tests exactly one set of names in our semantics. In $T_S^N[\ ]$, each process in the first parallel product sends $\overline{t}$ until it receives $\overline{x}$ and performs a step $\overline{t} \mid t \to 0$; each process in the second parallel product sends $\overline{t}$ when it receives $\overline{x}$; finally, $t.\overline{t}'$ forwards a message from t to t'. Hence, until P commits to S and provides $\prod_{x \in S} \overline{x}$, the process $T_S^N[P]$ can send the message $\overline{t}'$ For all P of sort N, we have $T_S^N[P] \Box\Downarrow_{t'}$ if and only if $S \notin P^\flat$. We conclude by definition of $\sqsubseteq_{\text{fair}}$. □

We now establish that our semantics corresponds to barbed coupled similarity:

Lemma 28 *In the local π-calculus, we have $\subseteq_\flat = \dot{\lesssim}$.*

PROOF. We successively check that $\subseteq_\flat$ refines the barbs, is a simulation, and meets the coupling condition. Assume $P \subseteq_\flat Q$.

- $P \Downarrow_x$ if and only if $x \in \bigcup P^\flat$ and, since $P^\flat \subseteq Q^\flat$, we also have $Q \Downarrow_x$.
- Since $P^\flat$ decreases with reductions, $\subseteq_\flat$ is trivially a simulation: if $P \to^* P'$, then $P' \subseteq_\flat P \subseteq_\flat Q'$.
- Using Remark 26, there is some $S \in P^\flat$. By hypothesis, $S \in Q^\flat$ and, for some process Q', we have $Q \to^* Q'$ and $Q'^\flat = \{S\} \subseteq P^\flat$, that is $P \supseteq_\flat Q'$. □

From Lemmas 27 and 28, we conclude that the precongruence of barbed coupled similarity yields fair testing, and is thus strictly coarser than barbed coupled precongruence.

Theorem 4 *In the local π-calculus, we have $\dot{\lesssim}^\circ = \sqsubseteq_{\text{fair}}$, $\dot{\lesssim}^\circ \subset \lesssim$, and thus $\dot{\lessapprox}^\circ = \simeq_{\text{fair}}$ and $\dot{\lessapprox}^\circ \subset \lessapprox$.*

7 Double-Barbed Bisimilarity

We now give a proof of $\dot{\approx}^\circ \subseteq \approx$ in the asynchronous π-calculus. Our proof holds for both the π-calculus and for the local π-calculus; it depends on the presence or absence of name matching only in Lemma 34, which handles both cases.

We rely on several encodings of values into the π-calculus. These standard continuation-passing-style encodings use only a deterministic fragment of the π-calculus, see, e.g., [30]. In the π-calculus, messages carry only names; hence, a process $\overline{x}\langle V\rangle$ that sends a message carrying the value V in the domain of the encoding is translated as $\nu u.(\langle\!\langle V \rangle\!\rangle_u \mid \overline{x}\langle u\rangle)$ where u is a fresh name and $\langle\!\langle V \rangle\!\rangle_u$ is a replicated input on u that receives continuations $\widetilde{c}$ and, depending on the structure of V, sends back (name-encoded) values $\widetilde{u}_i$ on one of those continuations $c_i \in \widetilde{c}$.

We first give an encoding for integers and their operations. Let u, u', v range over names representing integers in processes (with communication type $\iota \stackrel{\text{def}}{=} \mu\,\iota\,.\langle\langle\rangle, \langle\iota\rangle\rangle$) and let $n \geq 0$ represent integer constants. We use the encoding:

$$
\begin{aligned}
\langle\!\langle 0 \rangle\!\rangle_u &\stackrel{\text{def}}{=} !u(z,s).\overline{z} \\
\langle\!\langle v+1 \rangle\!\rangle_u &\stackrel{\text{def}}{=} !u(z,s).\overline{s}\langle v\rangle \\
\langle\!\langle n+1 \rangle\!\rangle_u &\stackrel{\text{def}}{=} \nu v.(\langle\!\langle n \rangle\!\rangle_v \mid \langle\!\langle v+1 \rangle\!\rangle_u) \\
\texttt{match}\ u\ \texttt{with}\ 0 \mapsto P\ \texttt{or}\ v+1 \mapsto Q &\stackrel{\text{def}}{=} \nu z,s.(\overline{u}\langle z,s\rangle \mid z.P \mid s(v).Q) \\
\texttt{if}\ u = u'\ \texttt{then}\ P\ \texttt{else}\ Q &\stackrel{\text{def}}{=} \\
\nu e.(\overline{e}\langle u,u'\rangle \mid !e(i,j).&\texttt{match}\ i\ j\ \texttt{with} \\
&\begin{cases}
0 & 0 & \mapsto P \\
0 & j'+1 & \mapsto Q \\
i'+1 & 0 & \mapsto Q \\
i'+1 & j'+1 & \mapsto \overline{e}\langle i', j'\rangle
\end{cases}
\end{aligned}
$$

In the definition above, the multiple matching is the usual shorthand for nested primitive matchings, and we assume that z, s, e, i, j, i', j' do not occur in P or Q.

We also use an injective function $[\![\ ]\!]$ from names $z \in \mathcal{N}$ to integers $[\![z]\!] \in \mathbb{N}$.

Next, we define a series of auxiliary equivalences:

Definition 29 *Let $(x_i)_{i\in\mathbb{N}}$ be a family of distinct nullary names. We let $\dot{\approx}_n$ be the largest symmetric bisimulation that refines the barbs $\Downarrow_{x_1}, \ldots, \Downarrow_{x_n}$, and let $\approx_n$ be the largest such bisimulation that is preserved by application of evaluation contexts.*

By construction, we have $\dot{\approx} \subseteq \dot{\approx}_n$, $\approx \subseteq \approx_n \subseteq \dot{\approx}^\circ_n$, and $\dot{\approx}^\circ \subseteq \dot{\approx}^\circ_n$, and the discriminating power of these n-barb equivalences increases with n. Obviously, $\dot{\approx}^\circ_0$ relates all processes. In the π-calculus, we have $\dot{\approx}^\circ_1 = \dot{\approx}^\circ_\exists$ and $\approx_n = \approx$ for any

$n \geq 1$. In addition, we are going to show that, for any $n \geq 2$, we have in fact $\dot{\approx}_n^\circ = \approx$. To this end, we focus on $\dot{\approx}_2$, and let x and y be the two nullary names associated with the barbs $\Downarrow_x$ and $\Downarrow_y$ refined by $\dot{\approx}_2$.

7.1 Some Equivalence Classes for $\dot{\approx}_2$

We first build a family of processes that are not $\dot{\approx}_2$-equivalent and retain this property by reduction. Informally, these processes represent infinitely many ways of hesitating between two messages in a branching semantics. The construction is general, and relies on an operator $\mathcal{S}(\cdot)$ that maps every set of processes $\mathcal{P}_i \subseteq \mathcal{P}$ to the set of its (strict, finite) internal sums:

Lemma 30 *Let $\mathcal{S}(\mathcal{P}_i) \stackrel{\text{def}}{=} \{\bigoplus_{P \in \mathcal{P}'} P \mid \mathcal{P}'$ is a finite subset of $\mathcal{P}_i$ and $|\mathcal{P}'| \geq 2\}$. For some given set of processes $\mathcal{P}_0 \subseteq \mathcal{P}$, let $\mathcal{P}_{n+1} = \mathcal{S}(\mathcal{P}_n)$ for $n \geq 0$ and $\mathcal{P}_\omega = \bigcup_{n \geq 0} \mathcal{P}_n$.*

We say that $\mathcal{S} \subseteq \mathcal{P}$ is $\mathcal{R}$-discrete when, for all $P, Q \in \mathcal{S}$, $P \mathcal{R} Q$ implies $P = Q$.

If $\mathcal{P}_0$ is $(\rightarrow^* \dot{\approx}_2)$-discrete, then (1) $\mathcal{P}_1$ is $(\rightarrow^* \dot{\approx}_2)$-discrete, and (2) $\mathcal{P}_\omega$ is $\dot{\approx}_2$-discrete.

PROOF. We first show that:

- *(3)* If $P, Q \in \mathcal{P}_0$, $P \rightarrow^* \dot{\approx}_2 R$, and $R \rightarrow^* \dot{\approx}_2 Q$, then $P = Q$.
  By bisimulation, we can compose the relations above and obtain $P \rightarrow^* \dot{\approx}_2 Q$.
  By hypothesis on $\mathcal{P}_0$ we obtain $P = Q$.
- *(4)* If $P \in \mathcal{P}_0$ and $R \in \mathcal{P}_1$, then we cannot have $P \rightarrow^* \dot{\approx}_2 R$.
  By construction of $\mathcal{P}_1$, we have $R \rightarrow^* \dot{\approx}_2 Q_i$ for at least two different $Q_i \in \mathcal{P}_0$, whereas, by *(3)*, $P \rightarrow^* \dot{\approx}_2 R$ yields $P = Q_i$ for all Q_i.

To prove *(1)*, consider $P, Q \in \mathcal{P}_1$ such that $P \rightarrow^* \dot{\approx}_2 Q$, that is, $P \rightarrow^* P'' \dot{\approx}_2 Q$ for some P''. Since P is an internal choice on some subset of $\mathcal{P}_0$ for $\dot{\approx}_2$, and *(4)* excludes $P' \rightarrow^* \dot{\approx}_2 P'' \dot{\approx}_2 Q$ for any $P' \in \mathcal{P}_0$, we actually have $P \dot{\approx}_2 P'' \dot{\approx}_2 Q$. Let $Q' \in \mathcal{P}_0$ be a summand of Q. By bisimulation, we have $P \rightarrow^* \dot{\approx}_2 Q'$. Since *(4)* excludes $Q' \dot{\approx}_2 P$, there exists $P' \in \mathcal{P}_0$ such that $P \rightarrow^* \dot{\approx}_2 P' \rightarrow^* \dot{\approx}_2 Q'$. By hypothesis on $\mathcal{P}_0$, we obtain $P' = Q'$, and thus Q' is also a summand of P. Symmetrically, every summand of P is a summand of Q, and finally $P = Q$.

To prove *(2)*, by induction on n, we show that every $\mathcal{P}_n$ is $\rightarrow^* \dot{\approx}_2$-discrete, that $P \in \mathcal{P}_n$, $Q \in \mathcal{P}_{n+m}$, and $P \rightarrow^* \dot{\approx}_2 Q$ imply $P = Q$, and that, in particular, $P, Q \in \mathcal{P}_\omega$ and $P \dot{\approx}_2 Q$ imply $P = Q$. □

We apply Lemma 30 to the set $\mathcal{P}_0 \stackrel{\text{def}}{=} \{ \mathtt{0}, \ \overline{x}, \ \overline{y} \}$. This set is clearly $(\rightarrow^* \dot{\approx}_2)$-discrete, since its processes have distinct barbs and no reductions. The size of each

layer $\mathcal{P}_n$ grows exponentially, and thus $\mathcal{P}_\omega$ contains infinitely many processes unrelated by $\dot{\approx}_2$. (Of course, $\dot{\approx}_2$ has more classes than those represented in $\mathcal{P}_\omega$, such as processes that can reach an infinite number of classes in $\mathcal{P}_\omega$.) Note that the construction also applies to the existential bisimilarities of Sections 5.1 and 6, but quickly converges. Starting from the set $\{\mathtt{0}, \overline{x}\}$, we obtain a third, unrelated process $\mathtt{0} \oplus \overline{x}$ at rank 1, then the construction yields no further classes.

The next lemma states that, thanks to the discriminating power of $\dot{\approx}_2$, a process can effectively pass any integer to its context by hesitating between the two exclusive barbs $\Downarrow_x$ and $\Downarrow_y$, without actually sending messages on x and y. To every integer, we associate a particular equivalence class of $\dot{\approx}_2$ in the hierarchy $\mathcal{P}_\omega$ (as depicted in Figure 1 below), then we write a process that receives an integer encoding and conveys its value by conforming to its characteristic class. Hence, the context $N[\cdot]$ transforms integer-indexed barbs $\overline{int}\langle n\rangle$ (where int is an ordinary name of the π-calculus) into the barbs $\Downarrow_x$ and $\Downarrow_y$.

Lemma 31 *There is a π-calculus evaluation context $N[\,]$ such that:*

(1) $N[\,]$ has sort $\{x, y\}$, binds $\{int\}$, and receives a single message on int.
(2) Let $N_n \stackrel{def}{=} N[\overline{int}\langle n\rangle]$. For all $n, m \in \mathbb{N}$, if $N_n \rightarrow^* \dot{\approx}_2 N_m$, then $n = m$.
(3) For all $n \in \mathbb{N}$, we have $N_n \not\dot{\approx}_2 N[\mathtt{0}]$.

PROOF. We program the evaluation context $N[\,]$ as follows, and we locate the derivatives of each N_n in the family $(\mathcal{P}_k)_{k<\omega}$ obtained from Lemma 30.

$$
\begin{aligned}
I &\stackrel{\text{def}}{=} !c(u,x,y,z).\mathtt{match}\ u\ \mathtt{with}\ 0 \mapsto \overline{x} \\
&\qquad\qquad\qquad\qquad \mathtt{or}\ v+1 \mapsto (\overline{c}\langle v,z,x,y\rangle \oplus \overline{c}\langle v,y,z,x\rangle) \\
J_u &\stackrel{\text{def}}{=} \overline{c}\langle u,x,y,z\rangle \oplus \overline{c}\langle u,z,x,y\rangle \oplus \overline{c}\langle u,y,z,x\rangle \\
N[\,] &\stackrel{\text{def}}{=} \nu\, int.([\,] \mid int(u).\nu c,z.(I \mid J_u))
\end{aligned}
$$

Intuitively, messages on c carry an integer loop index u and a permutation of the names x, y, and z; the replicated input I is used to iterate a binary internal choice from u to 1, whereas J_u is a single, initial, ternary internal choice.

Let ρ be the substitution that maps (x, y, z) to (z, x, y). Let σ range over $\rho^k, k \geq 0$. Let $Q_n^\sigma \stackrel{\text{def}}{=} \nu c,z.(I \mid \overline{c}\langle n, x\sigma, y\sigma, z\sigma\rangle)$, with $Q_n \stackrel{\text{def}}{=} Q_n^{Id}$. By construction, for any $n > 0$, we have:

$$Q_n^\sigma \approx Q_{n-1}^{\sigma\rho} \oplus Q_{n-1}^{\sigma\rho\rho} \tag{7}$$
$$N_n \approx Q_{n-1} \oplus Q_{n-1}^{\rho} \oplus Q_{n-1}^{\rho\rho} \tag{8}$$

The equivalence classes and their reductions are displayed in Figure 1, up to the permutation of Q_{2n+1} and $Q_{2n+1}^{\rho\rho}$ for $n \geq 0$.

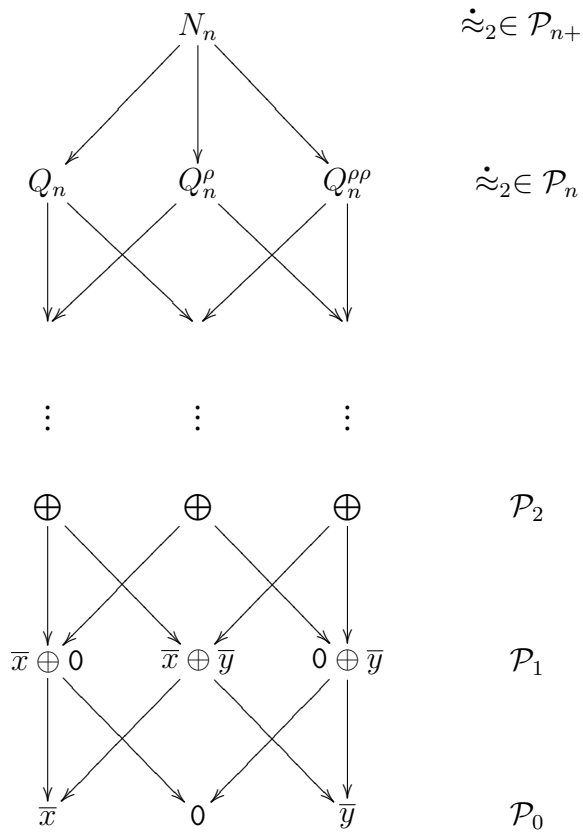


Fig. 1. Reduction classes for $N_n, n \geq 0$.

We say that $P \in \mathcal{P}_n$ is a *binary process* when either $n = 0$ or P is the sum of two distinct binary processes in $\mathcal{P}_{n-1}$. Binary processes are closed by reduction up to $\approx$.

By induction on n, we show that $Q_n \approx P_n$ for some binary process $P_n \in \mathcal{P}_n$. At rank 0, we have $Q_0 \approx \overline{x}$, $Q_0^\rho \approx 0$, and $Q_0^{\rho\rho} \approx \overline{y}$. For the inductive case, we apply (7) to our hypotheses, and the substitution ρ guarantees that the two summands are distinct. By composing this result with (8), we obtain $N_n \approx N'_n$ for some process $N'_n \in \mathcal{P}_{n+1}$ that is a ternary sum of binary processes.

Property *(1)* directly follows from our definitions. For property *(2)*, if $N_n \rightarrow^* \dot{\approx}_2 N_m$ then also $N'_n \approx \rightarrow^* \dot{\approx}_2 \approx N'_m$ for some ternary sums $N'_n \in \mathcal{P}_{n+1}$ and $N'_m \in \mathcal{P}_{m+1}$. Since $\approx \subseteq \dot{\approx}_2$ and $\dot{\approx}_2$ is a bisimulation, we have $N'_n \rightarrow^* \dot{\approx}_2 N'_m$. Either $N'_n \dot{\approx}_2 N'_m$, and thus $n = m$ by Lemma 30(2), or $N'_n \rightarrow^* \dot{\approx}_2 P_n \rightarrow^* \dot{\approx}_2 N'_m$ for some binary process P_n. The latter case implies that N'_m is also a binary process, and contradicts the construction of N'_m. Finally, property *(3)* follows from Lemma 30(2), since $N_n \approx N'_n \in \mathcal{P}_{n+1}$ and $N[0] \dot{\approx}_2 0 \in \mathcal{P}_0$. □

The next lemma uses this result to restrict the class of contexts being considered in congruence properties to contexts with at most two free nullary variables. (State-

ments *(2)* and *(3)* of the lemma are specifically used in the proof of Lemma 39.)

Lemma 32 *Let S be a finite set of names. Let $N[\ ]$ be the context of Lemma 31 for some int $\notin S$. There are evaluation contexts $F_S[\ ]$ of sort $\{int\}$ and $B_S \stackrel{def}{=} N[F_S[\ ]]$ such that, for all processes P and Q of sort S, we have:*

(1) If $B_S[P] \dot{\approx}_2 B_S[Q]$, then $P \dot{\approx} Q$.
(2) If $B_S[P] \to^* T \to^* \dot{\approx}_2 B_S[Q]$ then $P \to^* P'$ and $T \equiv B_S[P']$ for some P'.
(3) For some $k \in \mathbb{N}$, if $B_S[P] \to^* \dot{\approx}_2 N_n$, then $n < k$.

PROOF. Let $a, b \notin S$ be two nullary names. For all $z \in S \uplus \{a, b\}$, let $\widetilde{w}_z$ be a tuple of fresh names whose length matches the arity of z. To build $B_S[\ ]$, we use the additional terms:

$$\begin{aligned} X &\stackrel{\text{def}}{=} \textstyle\bigoplus_{z \in S \uplus \{a,b\}} z(\widetilde{w}_z).\overline{int}\langle [\![z]\!] \rangle \\ F_S[\ ] &\stackrel{\text{def}}{=} \nu S, a, b.\big([\ ] \mid \overline{a} \mid \overline{b} \mid X\big) \end{aligned}$$

(where $[\![\ ]\!]$ is our injective function from names to integers). By construction, for any P of sort S, we have $F_S[P]$ of sort $\{int\}$ and $B_S[P]$ of sort $\{x, y\}$. As soon as a message $\overline{int}\langle [\![z]\!] \rangle$ is sent by a derivative of X, the resulting process is bisimilar to $N_{[\![z]\!]}$, independently of the rest of the process enclosed in $N[\ ]$. For any P of sort S, we thus always have the reductions:

$$\begin{aligned} &B_S[P] \to^* B'_z[P] && \text{for any } z \in S \\ &B_S[P] \to^* B'_z[P] \dot{\approx}_2 N_{[\![z]\!]} && \text{for at least } z = a \text{ and } z = b \end{aligned}$$

where $B'_z[\ ]$ is obtained from $B_S[\ ]$ by choosing $z(\widetilde{w}_z).\overline{int}\langle [\![z]\!] \rangle$ in X.

We first show that, for all reductions $B_S[P] \to^* T$, there exists P' such that $P \to^* P'$ and one of the following holds:

- **(X)** $T \equiv B_S[P']$.
- **(z?)** For some $z \in S$, $T \equiv B'_z[P']$.
- **(z)** For some $z \in S \cup \{a, b\}$, $N_{[\![z]\!]} \to^* \dot{\approx}_2 T$ and $(P' \mid \overline{a} \mid \overline{b}) \downarrow_z$.

The proof is by induction on the length of the derivation, and case analysis on the first reduction step that reduces X. Before this step, $B_S[\ ]$ does not interact with P and we remain in case (X). After this step, if $z \in S$, the context $B'_z[\ ]$ interacts with P only if P sends a message on z and $z(\widetilde{w}_z).\overline{int}\langle [\![z]\!] \rangle$ receives it. Until this step occurs, we remain in case (z?). When it occurs, we arrive in case (z). After the step that leaves (X), if $z = a$ or $z = b$ is chosen, communication on z can always occur independently of P, so we are already in case (z).

We now establish property *(3)* of the lemma. Let P be a process of sort S such that $B_S[P] \rightarrow^* T \mathrel{\dot{\approx}}_2 N_n$. In case (X) and case (z?) with $P' \Downarrow_z$, we have $T \rightarrow^* \mathrel{\dot{\approx}}_2 N_{[\![z]\!]}$ for some $z \in S \cup \{a, b\}$, and then $n = [\![z]\!]$ by Lemma 31(2). In case (z), we have $N_{[\![z]\!]} \rightarrow^* \mathrel{\dot{\approx}}_2 T$ and similarly $n = [\![z]\!]$. In case (z?) with $P' \not\Downarrow_z$, we have $T \mathrel{\dot{\approx}}_2 N[\mathsf{0}]$, which contradicts Lemma 31(3). Thus, n is bounded by the largest $[\![z]\!]$ for $z \in S \cup \{a, b\}$.

To prove property *(2)*, assume $B_S[P] \rightarrow^* T \rightarrow^* \mathrel{\dot{\approx}}_2 B_S[Q]$. We rely on the case analysis above for the reductions $B_S[P] \rightarrow^* T$. To conclude, we show that we are always in case (X). Otherwise, let z be the name chosen in $B_S[P] \rightarrow^* B'_z[P'] \rightarrow^* T$. We have $B_S[Q] \rightarrow^* \mathrel{\dot{\approx}}_2 N_n$ for at least one value $n \neq [\![z]\!]$ (either $n = [\![a]\!]$ or $n = [\![b]\!]$). By bisimulation, we obtain $B'_z[P'] \rightarrow^* T \rightarrow^* T' \mathrel{\dot{\approx}}_2 N_n$.

We use our case analysis again, for the reductions $B_S[P] \rightarrow^* B'_z[P'] \rightarrow^* T'$. In case (z), we have $N_{[\![z]\!]} \rightarrow^* \mathrel{\dot{\approx}}_2 T'$, hence $N_{[\![z]\!]} \rightarrow^* \mathrel{\dot{\approx}}_2 N_n$ and, by Lemma 31, we obtain the contradiction $[\![z]\!] = n$. In case (z?), we have $T' \equiv B'_z[P'']$, with two subcases. If $P'' \Downarrow_z$, then $T' \rightarrow^* \mathrel{\dot{\approx}}_2 N_{[\![z]\!]}$ and $T' \mathrel{\dot{\approx}}_2 N_n$ also yields the contradiction $[\![z]\!] = n$. Otherwise ($P'' \not\Downarrow_z$), we have $T' \mathrel{\dot{\approx}}_2 N[\mathsf{0}]$ and thus $N_n \mathrel{\dot{\approx}}_2 N[\mathsf{0}]$, which contradicts Lemma 31(3).

To prove property *(1)*, let $\mathcal{R}$ be the relation that contains all pairs (P, Q) with $\mathrm{fv}(P) \cup \mathrm{fv}(Q) \subseteq S$ and $B_S[P] \mathrel{\dot{\approx}}_2 B_S[Q]$. We show that $\mathcal{R}$ is a barbed bisimulation, and thus $\mathcal{R} \subseteq \mathrel{\dot{\approx}}$. For any $P \mathrel{\mathcal{R}} Q$:

Barbs Let $z \in S$. If $P \Downarrow_z$, then $B_S[P] \rightarrow^* \mathrel{\dot{\approx}}_2 N_{[\![z]\!]}$. By hypothesis, $B_S[P] \mathrel{\dot{\approx}}_2 B_S[Q]$, hence, by bisimulation, we obtain $B_S[Q] \rightarrow^* \mathrel{\dot{\approx}}_2 N_{[\![z]\!]}$. Using the case analysis above, $B_S[Q] \rightarrow^* \mathrel{\dot{\approx}}_2 N_{[\![z]\!]}$ yields $Q \Downarrow_z$.

Bisimulation: If $P \rightarrow^* P'$, then $B_S[P] \rightarrow^* B_S[P']$ and, since $B_S[P] \mathrel{\dot{\approx}}_2 B_S[Q]$, we have $B_S[Q] \rightarrow^* T$ with $B_S[P'] \mathrel{\dot{\approx}}_2 T$. Using property *(2)*, we obtain reductions $Q \rightarrow^* Q'$ such that $T \equiv B_S[Q']$, hence $P' \mathrel{\mathcal{R}} Q'$. □

7.2 π-calculus Interpreters

For a given finite sort S, we define an interpreter process R_u with free variables $S \uplus \{u\}$ that interprets u as the integer-coded representation of a π-calculus process. Whenever u encodes a process P with sort S, the interpreter behaves like P up to labeled bisimilarity ($R_u \approx_l P$). As opposed to most lemmas in Section 7, the actual definition of the interpreter is sensitive to small variations in the calculus, including its type system. We first give a finite interpreter for processes that use only replicated input and a finite number of channel types, then use preliminary internal encodings to extend the interpreter to arbitrary processes.

Definition 33 *Let Σ be a finite set of types; we say that a process P is* Σ-proper *when (1) all free and bound names of P are typed in Σ, and (2) for all subterms*

of P of the form $!Q$, Q is of the form $x(\widetilde{y}).Q'$.

Next, we give an integer encoding for the syntax of Σ-proper processes. We write $[\![P]\!]$ for the integer that represents the (typed) abstract syntax tree for P, as defined in Section 2. The encoding relies on our injective function from names x to integers $[\![x]\!]$ and on an arbitrary injective function from the types $\sigma \in \Sigma$ to integers $[\![\sigma]\!]$.

The process syntax encoding is basically a Gödel numbering with type indexes inserted for input, output and restriction constructs. We use an $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ bijection, defined by $\eta(j,k) \stackrel{\text{def}}{=} 2^j(2k+1) - 1$; we also use $\eta(j_1, \ldots, j_n)$ as shorthand for $\eta(j_1, \eta(j_2, \ldots, \eta(j_{n-1}, j_n)\ldots))$. If the channel name x has type $\sigma \in \Sigma$ in the context of the translation, we take:

$$
\begin{aligned}
[\![\mathtt{0}]\!] &\stackrel{\text{def}}{=} 0 \\
[\![P \mid Q]\!] &\stackrel{\text{def}}{=} \eta(1, [\![P]\!], [\![Q]\!]) \\
[\![\nu x : \sigma.P]\!] &\stackrel{\text{def}}{=} \eta(5[\![\sigma]\!] + 2, [\![x]\!], [\![P]\!]) \\
[\![\overline{x}\langle\widetilde{y}\rangle]\!] &\stackrel{\text{def}}{=} \eta(5[\![\sigma]\!] + 3, [\![x]\!], \widetilde{[\![y]\!]}) \\
[\![x(\widetilde{y}).P]\!] &\stackrel{\text{def}}{=} \eta(5[\![\sigma]\!] + 4, [\![x]\!], \widetilde{[\![y]\!]}, [\![P]\!]) \\
[\![!x(\widetilde{y}).P]\!] &\stackrel{\text{def}}{=} \eta(5[\![\sigma]\!] + 5, [\![x]\!], \widetilde{[\![y]\!]}, [\![P]\!]) \\
[\![[x = x']P]\!] &\stackrel{\text{def}}{=} \eta(5[\![\sigma]\!] + 6, [\![x]\!], [\![x']\!], [\![P]\!])
\end{aligned}
$$

Since $[\![\mathtt{0} \mid P]\!] > [\![P]\!]$, for any k and P, there exists some $Q \equiv P$ with $[\![Q]\!] > k$.

We also define a pattern-matching syntax of processes for inverting η:

$$
\begin{aligned}
&\mathtt{match}\ u\ \mathtt{with}\ \eta(e,m)\ \mathtt{in}\ P \stackrel{\text{def}}{=} \\
&\qquad \nu s.(\overline{s}\langle u,u,u,u\rangle \mid\ !s(i,m,j,e).\mathtt{match}\ i\ m\ j\ e\ \mathtt{with} \\
&\qquad\quad \begin{cases}
0 & \_ & \_ & \_ & \mapsto P \\
1 & m'+1 & \_ & \_ & \mapsto \overline{s}\langle m', m', j, j\rangle \\
i'+2 & m'+1 & j'+1 & e'+2 & \mapsto \overline{s}\langle i', m', j', e'\rangle \\
\_ & \_ & \_ & \_ & \mapsto \mathtt{0})
\end{cases} \\
&\mathtt{match}\ u\ \mathtt{with}\ \eta(v_1, \ldots, v_n)\ \mathtt{in}\ P \stackrel{\text{def}}{=} \\
&\qquad \mathtt{match}\ u\ \mathtt{with}\ \eta(v_1, u')\ \mathtt{in}\ \mathtt{match}\ u'\ \mathtt{with}\ \eta(v_2, \ldots, v_n)\ \mathtt{in}\ P
\end{aligned}
$$

where we assume that s, u', i, j do not occur in P. Whenever u encodes $n \in \mathbb{N}$, the pattern matching completes and triggers P binding (encodings of) $e, m \in \mathbb{N}$ such that $\eta(e,m) = n$ (and, respectively, binding $v_1, \ldots v_n \in \mathbb{N}$ such that $\eta(v_1, \ldots, v_n) = n$). The clause '$\_ \mapsto \mathtt{0}$' is never selected. The correctness of the decoding of $\eta(e,m)$ follows from the invariants $2^{e-i}(2m+1-i) - 1 = n$, $i + j = m + e$, $0 \le i \le m \le j$, and $0 \le i \le e$. Its termination follows from decreasing m's.

Finally, we define a process encoding of finite association tables ρ from integers to names typed in Σ, which we will refer to as Σ-*tables*. (The domain of our Σ-tables will consist of images $[\![z]\!]$ of names z under the $[\![\ ]\!]$ injection.) A Σ-table is either the empty table $\emptyset$, or the overriding extension $\rho\{y : \tau \,/\, [\![z]\!]\}$ of another table ρ. The general form of a Σ-table is thus $\emptyset\{\widetilde{y : \tau \,/\, [\![z]\!]}\}$. In the process $P\rho$, we identify ρ with its implied substitution—that is, $P(\emptyset\{\widetilde{y : \tau \,/\, [\![z]\!]}\}) \stackrel{\text{def}}{=} P\{\widetilde{y/z}\}$.

$$
\begin{aligned}
\langle\!\langle \emptyset \rangle\!\rangle_r &\stackrel{\text{def}}{=} \mathtt{0} \\
\langle\!\langle \rho\{y : \tau \,/\, u_y\} \rangle\!\rangle_r &\stackrel{\text{def}}{=} \nu r'.(\langle\!\langle \rho \rangle\!\rangle_{r'} \mid \langle\!\langle r'\{y : \tau \,/\, u_y\} \rangle\!\rangle_r) \\
\langle\!\langle r'\{y : \tau \,/\, u_y\} \rangle\!\rangle_r &\stackrel{\text{def}}{=} !r(u, c).\mathtt{if}\ u = u_y\ \mathtt{then}\ \nu\widetilde{z_\sigma : \sigma}.\overline{c}\langle\widetilde{z_\sigma}\{{}^{y}/_{z_\tau}\}\rangle\ \mathtt{else}\ \overline{r'}\langle u, c\rangle
\end{aligned}
$$

where $\widetilde{z_\sigma}$ is a tuple of fresh names indexed by the types of Σ, and where $\widetilde{z_\sigma}\{{}^{x}/_{z_\tau}\}$ is $\widetilde{z_\sigma}$ with the name at index τ replaced by x. We need this tuple to ensure that the encoding is well-typed; the type of Σ-tables is thus $\langle \iota, \langle\widetilde{\Sigma}\rangle\rangle$, where ι is the type of integer encodings. We also give a corresponding `let`-syntax for accessing Σ-tables.

$$
\begin{aligned}
\mathtt{let}\ x : \tau = r[m]\ \mathtt{in}\ P &\stackrel{\text{def}}{=} \nu c.(\overline{r}\langle m, c\rangle \mid c(\widetilde{z_\sigma}\{{}^{x}/_{z_\tau}\}).P) \\
\mathtt{let}\ x_0, \widetilde{x} : \tau_0, \widetilde{\tau} = r[m_0, \widetilde{m}]\ \mathtt{in}\ P &\stackrel{\text{def}}{=} \mathtt{let}\ x_0 : \tau_0 = r[m_0]\ \mathtt{in}\ \mathtt{let}\ \widetilde{x} : \widetilde{\tau} = r[\widetilde{m}]\ \mathtt{in}\ P
\end{aligned}
$$

The next lemma relates processes P to the interpretation of their representation $[\![P]\!]$. As long as the interpreter can be finitely defined, the result is not surprising, since the π-calculus has sufficient expressive power. In particular, similar interpreters should be definable for most variants of the π-calculus.

Lemma 34 (Finite Interpreter) *In the π-calculus, with or without name matching, let Σ be a finite set of types and let S be a finite set of typed names with types in Σ. There is a process R_u of sort $S \uplus \{u\}$ such that, for every Σ-proper process P with sort S, we have $\nu u.(\langle\!\langle [\![P]\!] \rangle\!\rangle_u \mid R_u) \approx_l P$.*

PROOF. Let ρ_S be the finite table $\emptyset\{\widetilde{y : \tau_y \,/\, [\![y]\!]}\}$, where $y : \tau_y$ ranges over S. We define the interpreter $R_u \stackrel{\text{def}}{=} R_u(\rho_S)$ in Figure 2, using the encodings for processes, names, integers, types, and Σ-tables specified above, with an auxiliary replicated input D_ε that recursively performs pattern matching on the process coded by u in the Σ-table coded by r. The interpreter closely follows the syntax and types for processes. The last series of clauses is present only when the source π-calculus has a name matching prefix, and is implemented using the same prefix.

In R_u, reduction steps are either steps in strong correspondence with those of the source process, on the same channel names and with the same arguments, or bookkeeping steps: steps for the encodings, and reductions on ε. We write $\rightarrow_d$ for those

$$R_u(\rho) \stackrel{\text{def}}{=} \nu\varepsilon.(D_\varepsilon \mid \nu r.(\langle\!\langle \rho \rangle\!\rangle_r \mid \overline{\varepsilon}\langle u, r\rangle))$$

$$D_\varepsilon \stackrel{\text{def}}{=} !\varepsilon(u,r).\texttt{match } u \texttt{ with } \eta(t,u') \texttt{ in match } t \texttt{ with}$$

$$\begin{cases}
\quad 0 \mapsto \texttt{0} \\
\quad 1 \mapsto \texttt{match } u' \texttt{ with } \eta(u_1,u_2) \texttt{ in } \overline{\varepsilon}\langle u_1, r\rangle \mid \overline{\varepsilon}\langle u_2, r\rangle \\
\text{For every } \sigma = \langle\widetilde{\tau}\rangle \in \Sigma\text{:} \\
5[\![\sigma]\!]+2 \mapsto \texttt{match } u' \texttt{ with } \eta(m_x,u_1) \texttt{ in } \\
\qquad \nu x, r'.(\langle\!\langle r\{x : \sigma \,/\, m_x\}\rangle\!\rangle_{r'} \mid \overline{\varepsilon}\langle u_1, r'\rangle) \\
5[\![\sigma]\!]+3 \mapsto \texttt{match } u' \texttt{ with } \eta(m_x,\widetilde{m_y}) \texttt{ in let } x, \widetilde{y} : \sigma, \widetilde{\tau} = r[m_x, \widetilde{m_y}] \texttt{ in } \\
\qquad \overline{x}\langle\widetilde{y}\rangle \\
5[\![\sigma]\!]+4 \mapsto \texttt{match } u' \texttt{ with } \eta(m_x,\widetilde{m_y},u_1) \texttt{ in let } x : \sigma = r[m_x] \texttt{ in } \\
\qquad x(\widetilde{y}).\nu r'.(\langle\!\langle r\{y : \tau \,/\, m_y\}\rangle\!\rangle_{r'} \mid \overline{\varepsilon}\langle u_1, r'\rangle) \\
5[\![\sigma]\!]+5 \mapsto \texttt{match } u' \texttt{ with } \eta(m_x,\widetilde{m_y},u_1) \texttt{ in let } x : \sigma = r[m_x] \texttt{ in } \\
\qquad !x(\widetilde{y}).\nu r'.(\langle\!\langle r\{y : \tau \,/\, m_y\}\rangle\!\rangle_{r'} \mid \overline{\varepsilon}\langle u_1, r'\rangle) \\
\text{Only for the } \pi\text{-calculus with matching:} \\
5[\![\sigma]\!]+6 \mapsto \texttt{match } u' \texttt{ with } \eta(m,m',u_1) \texttt{ in let } x, x' : \sigma, \sigma = r[m,m'] \texttt{ in } \\
\qquad [x = x']\overline{\varepsilon}\langle u_1, r\rangle
\end{cases}$$

Fig. 2. Finite Interpreter

bookkeeping reduction steps. These bookkeeping steps are deterministic and normalizing : for any Σ-proper process P and Σ-table ρ, the process

$$R(P,\rho) \stackrel{\text{def}}{=} \nu u(\langle\!\langle [\![P]\!] \rangle\!\rangle_u \mid R_u(\rho))$$

has a $\rightarrow_d$-normal form, which is unique up to $\equiv$. Specifically, we have

$$\begin{array}{rcll} P\rho & \equiv & \nu\widetilde{y}.\,(I \mid \prod G_i.P_i\rho_i) & \\ R(P,\rho) & \rightarrow_d^* \sim & \nu\widetilde{y}.\,(I \mid \prod G_i.R(P_i,\rho_i)) & \text{in } \rightarrow_d\text{-normal form} \end{array}$$

where I is a product of Σ-proper output terms, each G_i is a guard (either input, replicated input, or matching), each P_i is a subterm of P (prior to α-conversion), and ρ_i is the Σ-table representing the substitution applied to P_i (which may perform α-conversion). We use the strong bisimulation in the second equation to replicate or discard D_ε and representations of integers and tables, and move these under guards. Note that the right-hand sides of both equations are unique up to structural equivalence.

Structural equivalence in the source process corresponds only to labeled bisimilarity in the interpreter—in particular, α-conversion may cause additional reductions on integer indices in the interpreter. We avoid this problem by using the normal forms; we let

$$\begin{aligned}\mathcal{R} \stackrel{\text{def}}{=} \{(P,Q) \mid\ & P \equiv \nu\widetilde{y}.\,(I \mid \textstyle\prod G_i.P_i\rho_i),\, Q \rightarrow_d^* \sim \nu\widetilde{y}.\,(I \mid \textstyle\prod G_i.R(P_i,\rho_i)) \\ & \text{for some names } \widetilde{y}\text{, guards } G_i\text{, some product of outputs } I\text{,} \\ & \text{and some } \Sigma\text{-proper process } P_i \text{ and } \Sigma\text{-tables } \rho_i \\ & \text{such that the sort of } P_i \text{ is included in the domain of } \rho_i\ \}\end{aligned}$$

(Note that there is no type restriction on I.) We prove that $\mathcal{R}$ is a labeled bisimulation : if $P\ \mathcal{R}\ Q$, then

- Asynchronous input and output steps of P and Q match trivially, since they only affect the I and $\widetilde{y}$ components, which are identical in the normal forms of P and Q, and bookkeeping reductions in Q can only extend I and $\widetilde{y}$.
- If $P \rightarrow P'$, we must have $I \equiv I_M \mid M$ for some I_M, M, and $(M \mid G_j.P_j\rho_j) \rightarrow P_j\rho_j\rho'$ for some j and substitution ρ', such that

  $$P' \equiv \nu\widetilde{y}\left(I_M \mid (P_j\rho_j)\rho' \mid \textstyle\prod_{i\neq j} G_i.P_i\rho_i\right)$$

  where G_j is either an input or a name matching (with $M = \mathtt{0}$ if G_j is a matching). We then have

  $$Q \rightarrow_d^* \rightarrow Q' \ \sim\ \nu\widetilde{y}.\left(I_M \mid R(P_j,\rho_j\rho') \mid \textstyle\prod_{i\neq j} G_i.R(P_i,\rho_i)\right)$$

  and we can further extend this computation with the bookkeeping steps that normalize $R(P_j,\rho_j\rho')$:

  $$Q' \rightarrow_d^* \sim \nu\widetilde{y},\widetilde{y}'.\left(I_M \mid I' \mid (\textstyle\prod G'_k.R(P'_k,\rho'_k)) \mid \left(\textstyle\prod_{i\neq j} G_i.R(P_i,\rho_i)\right)\right)$$

  Since the condition on P_j and ρ_j implies that $(P_j\rho_j)\rho' = P_j(\rho_j\rho')$ we then have $P' \equiv \nu\widetilde{y},\widetilde{y}'.\left(I_M \mid I' \mid (\prod G'_k.P'_k\rho'_k) \mid \left(\prod_{i\neq j} G_i.P_i\rho_i\right)\right)$, hence $P'\ \mathcal{R}\ Q'$.
- if $Q \rightarrow Q'$, then either $Q \rightarrow_d Q'$, and then $P\ \mathcal{R}\ Q'$, or, as above, there are some M, I_M, j, ρ' such that $Q' \rightarrow_d^* Q'' \sim \nu\widetilde{y}.\,(I_M \mid R(P_j,\rho_j\rho') \mid \prod G_i.R(P_i,\rho_i))$; then we have $P \rightarrow P' \stackrel{\text{def}}{=} \nu\widetilde{y}\,(I_M \mid (P_j\rho_j)\rho' \mid \prod G_i.P_i\rho_i)$ and, as above, considering the normal forms of Q'' and P' gives us $P'\ \mathcal{R}\ Q'$.

The main result follows from the fact that $P = P\rho_S\ \mathcal{R}\ R(P,\rho_S)$. □

We now need to show that our interpreter can emulate *all* processes of sort S, not just Σ-proper ones. Because the interpreter must be a finite process, some preliminary encoding is needed to eliminate arbitrarily-large syntactic constructs which might occur in the process to be interpreted. The problem occurs for the channels

that are never extruded. These channels can have arbitrary types, unrelated to Σ, and arbitrarily large arities. Since these channels are internal to the source process, we can use a structural, type-driven translation that implements communication on channels of these unrelated types with a series of communications on channels of some uniform type, in the spirit of the encoding from the polyadic π-calculus to its monadic subset (see, e.g., [30]). The correctness of the encoding rests on the following notion:

Definition 35 *A set of channel types Σ is closed under decomposition when, for each $\sigma \in \Sigma$, if $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$, then also $\sigma_1, \ldots, \sigma_n \in \Sigma$ (up to unfolding).*

With the simple type system given in Section 2, any finite set of type Σ has a finite smallest superset $D(\Sigma)$ that is closed under decomposition.

Lemma 36 *Let Σ be a finite set of types. There is a finite set of types $F(\Sigma) \supseteq \Sigma$ such that, for any process P whose free names are typed in $F(\Sigma)$, there exists P^o whose names are all typed in $F(\Sigma)$ with $P \approx_l P^o$.*

PROOF. We take $F(\Sigma) \stackrel{\text{def}}{=} D(\Sigma) \cup \{o\}$, where $o \stackrel{\text{def}}{=} \mu o.\langle \widetilde{D(\Sigma) \cup \{o\}}, o \rangle$. Note that $F(\Sigma)$ is closed under decomposition. We define a translation $(\cdot)^o$ on typed terms, setting for types $\sigma^o \stackrel{\text{def}}{=} \sigma$ when $\sigma \in F(\Sigma)$ and $\sigma^o \stackrel{\text{def}}{=} o$ when $\sigma \notin F(\Sigma)$.

For processes, the translation is compositional and type-driven. In the rules below, the tuple index σ in $\widetilde{z_\sigma}$ ranges over $F(\Sigma)$, while the i in $\widetilde{\tau_i}$, $\widetilde{u_i}$, $\widetilde{y_i}$, ranges over $\{1, \ldots, n\}$. (We assume that all names introduced in the translation are fresh.) The top two rules apply when the type of x is in $F(\Sigma)$; the next two rules apply when the type of y_0 is $\langle \widetilde{\tau_i} \rangle \notin F(\Sigma)$.

$$
\begin{array}{rcl}
(\overline{x}\langle \widetilde{u_i} \rangle)^o & \stackrel{\text{def}}{=} & \overline{x}\langle \widetilde{u_i} \rangle \\
(x(\widetilde{u_i}).P)^o & \stackrel{\text{def}}{=} & x(\widetilde{u_i}).P^o \\
(\overline{y_0}\langle \widetilde{u_i} \rangle)^o & \stackrel{\text{def}}{=} & \widetilde{\nu z_\sigma : \sigma}.\widetilde{\nu y_i : o}.\ \prod_{i=1}^{n}\ \overline{y_{i-1}}\langle \widetilde{z_\sigma}\{u_i/z_{\tau_i{}^o}\}, y_i \rangle \\
(y_0(\widetilde{u_i}).P)^o & \stackrel{\text{def}}{=} & y_0(\widetilde{z_\sigma}\{u_1/z_{\tau_1{}^o}\}, y_1). \ldots y_{n-1}(\widetilde{z_\sigma}\{u_n/z_{\tau_n{}^o}\}, y_n).P^o
\end{array}
$$

$$
\mathsf{0}^o \stackrel{\text{def}}{=} \mathsf{0} \qquad (P \mid Q)^o \stackrel{\text{def}}{=} P^o \mid Q^o \qquad (!P)^o \stackrel{\text{def}}{=} !P^o
$$

$$
(\nu z : \tau.P)^o \stackrel{\text{def}}{=} \nu z : \tau^o.P^o \qquad ([x = x']P)^o \stackrel{\text{def}}{=} [x = x']P^o
$$

The translation leaves any name that may be exchanged with the environment unchanged, and changes the type of some local names to reflect the use of a generic communication protocol.

To prove the correctness of the translation up to labeled bisimilarity, we show that the relation containing the pairs $(I \mid P, I \mid P^o)$ for all products of outputs I, and all P

whose free variables are all typed in $F(\sigma)$, is a labeled bisimilarity up to expansion [42]. Transitions in the translated process are either in direct correspondence with transitions in the source process, or additional internal steps on local names introduced by the encoding of output; these internal steps are deterministic. Using labeled expansion, we can perform all these additional steps immediately after any internal step on an encoded channel, and obtain the translation of the resulting source process after the internal step. Note that the closure property of $F(\Sigma)$ ensures that outputs in I can only interact with P or P^o when they are typed in $F(\Sigma)$, and conversely that outputs of P or P^o remain typed in $F(\Sigma)$. □

Using the expanded type set $F(\Sigma)$ of Lemma 36 in Lemma 34, the identity $!P \approx_l \nu z.(\overline{z}\,|!z.(P\,|\,\overline{z}))$ to replace general replication with replicated input, and the translation P^o to eliminate types outside $F(\Sigma)$, we obtain a universal interpreter:

Corollary 37 (Interpreter) *In the π-calculus, with or without name matching, let S be a finite set of typed names. There is a process R_u of sort $S \uplus \{u\}$ such that, for every process P of sort S, there exists a process Q such that $\nu u.(\langle\!\langle [\![Q]\!] \rangle\!\rangle_u \,|\, R_u) \approx_l Q \approx_l P$.*

While our interpreter may be adapted to various type systems (e.g., Lemma 36 may be weakened for the π-calculus with an infinite system of variable sorts, where $D(\Sigma)$ can be infinite), its existence is not always guaranteed. For instance, in the join calculus with polymorphism à la ML [18], the interpreter can be adapted to polymorphic types but, surprisingly, there is no finite interpreter if we also add name matching. In that setting, we still have $\approx \;=\; \approx_l$ but we cannot prove an equivalent of Theorem 1. On the contrary:

Lemma 38 *In the join calculus with both polymorphic types and name matching, labeled bisimilarity is strictly finer than the congruence of barbed bisimilarity: $\approx_l \subset \dot{\approx}^\circ$.*

PROOF. We give a counter-example in the join calculus, in the spirit of Brookes' counter-example between limit bisimulations and bisimilarity.

For any $n \in \mathbb{N}$, we let P_n be a process that performs a series of tests on a polymorphic name $\overline{f} : \forall\alpha.\langle \mathtt{Int}, \langle\langle\alpha\rangle\rangle\rangle$ encoding a function (in continuation passing style, cf. [18]). After an initial call to $f\langle 0, c\rangle$ returns, P_n successively calls $f\langle i, c\rangle$ *twice* for each $i \in \mathbb{N}$, and tests, if $i < n$, that (1) both calls return the *same* name v_i ; and (2) $v_i \neq v_j$ for any $j < i$. If any test fails, a single t is emitted; otherwise, if f passes all tests, P_n is nondeterministic, and may or may not emit t. Nothing happens if the initial call does not return. We also let P_ω be a process that performs the same calls to f, but 'fails' and emits a single t irrespective of the results (as long as the initial call returns). For instance, we can use the processes

$$
\begin{array}{l}
P_n \stackrel{\text{def}}{=} \texttt{def}\ x\langle\rangle \mid y\langle\rangle \triangleright 0 \wedge x\langle\rangle \mid z\langle\rangle \triangleright t\langle\rangle\ \texttt{in} \\
\quad \texttt{def}\ e\langle i, d\rangle \triangleright \\
\quad\quad \texttt{def}\ c\langle v\rangle \mid c'\langle v'\rangle \mid c''\langle\rangle \triangleright \\
\quad\quad\quad [i < n][v = v']d\langle v\rangle \mid \\
\quad\quad\quad \texttt{def}\ d'\langle u\rangle \triangleright\ d\langle u\rangle \mid [u = v]x\langle\rangle\ \texttt{in}\ e\langle i + 1, d'\rangle\ \texttt{in} \\
\quad\quad f\langle i, c\rangle \mid f\langle i, c'\rangle \mid c''\langle\rangle \mid [i < n]x\langle\rangle\ \texttt{in} \\
\quad \texttt{def}\ c\langle v\rangle \triangleright (z\langle\rangle \mid \texttt{def}\ d\langle v\rangle \triangleright y\langle\rangle\ \texttt{in}\ e\langle 0, d\rangle)\ \ \texttt{in}\ f\langle 0, c\rangle
\end{array}
$$

$$
\begin{array}{l}
P_\omega \stackrel{\text{def}}{=} \texttt{def}\ e\langle i\rangle \triangleright \\
\quad\quad \texttt{def}\ c\langle v\rangle \mid c'\langle v'\rangle \mid c''\langle\rangle \triangleright e\langle i + 1\rangle\ \texttt{in} \\
\quad\quad f\langle i, c\rangle \mid f\langle i, c'\rangle \mid c''\langle\rangle\ \texttt{in} \\
\quad \texttt{def}\ c\langle v\rangle \triangleright t\langle\rangle \mid e\langle 0\rangle\ \texttt{in}\ f\langle 0, c\rangle
\end{array}
$$

The π-calculus equivalent of the two join definitions that involve some synchronization would be here $!y.x \,|!z.x.\bar{t}$, and $!c''.c(v).c'(v').[\,]$, respectively. The other join definitions are equivalent to replicated inputs; in all cases, the `def` implies scope restriction. The integer operations (comparison prefix, zero, and increment) can be replaced with standard encodings.

For any $n \in \mathbb{N}$, if f honors the initial call, the process P_n can lose the ability to emit t ($P_n \not\Downarrow_t$) if and only if f passes all tests at rank n. Conversely, P_ω emits t as soon as the initial call returns.

For a given $n \in \mathbb{N}$, it is straightforward to write a context $C_n[\,]$ that defines a function f_n passing all tests for $i \leq n$ (and does not bind t). Conversely, in the case $C[P_n]$ can pass all tests, the context $C[\,]$ must have n different names $(v_i)_{i<n}$ to return. Since each name must be returned twice, these names cannot be created under a join pattern that defines f. By definition of the generalization criterion, names with a polymorphic type cannot be received in a join pattern that defines f. Hence, these returned values must already be defined in $C[\,]$, and the size of $C[\,]$ grows with n.

We now compare the processes $S_1 = \bigoplus_{n<\omega} P_n$ and $S_2 = \bigoplus_{n\leq\omega} P_n$, noting that if $S_i \rightarrow^* P$, then either $P \approx S_i$, or $P \approx P_n$ (with $n < \omega$ if $i = 1$), as the P_n have no transitions without a definition for f.

$S_1 \not\approx_l S_2$**:** the reduction $S_2 \rightarrow P_\omega$ cannot be matched by S_1. In the case $S_1 \rightarrow^* \approx P_n$, we add the context $C_n[\,]$; the process $C_n[P_n]$ can perform transitions and reach a state where it has lost its barb on t, while $C_n[P_\omega]$ cannot. Yet, $S_1 \not\approx_l P_\omega$ either, because no reduction $S_1 \rightarrow^* P_n$ can be matched by P_ω, for the same reason.

$S_1 \dot{\approx}^\circ S_2$**:** any given context $C[\,]$ can perform tests only at a bounded depth, hence there is $n \in \mathbb{N}$ such that, for any $m \geq n$, we have $C[P_m] \dot{\approx} C[P_\omega]$. □

We are now ready to prove $\dot{\approx}_2^\circ \subseteq \approx$. We build a single context $U_S[\ ]$ that has essentially the discriminating power of all contexts. We call this context a universal context.

Lemma 39 (Universal Context) *For all finite sets of typed names S such that $x, y \notin S$, there is an evaluation context $U_S[\ ]$ such that the relation*

$$\phi_S \stackrel{def}{=} \{(P, Q) \mid \mathrm{fv}(P) \cup \mathrm{fv}(Q) \subseteq S \text{ and } U_S[P] \dot{\approx}_2 U_S[Q]\}$$

has the following properties:

(1) Let $C[\ ]$ be an evaluation context such that $\mathrm{fv}(C[P]) \subseteq \{x, y\}$ for any P with $\mathrm{fv}(P) \subseteq S$. For all P and Q, if $P\ \phi_S\ Q$, then $C[P] \dot{\approx}_2 C[Q]$.
(2) Let σ range over injective substitutions on names. The relation $\phi \stackrel{def}{=} \{(P\sigma, Q\sigma) \mid \exists S.P\ \phi_S\ Q\}$ is a congruence and a barbed bisimulation, hence $\phi_S \subseteq \approx$.

PROOF. Let $B[\ ]$ be the evaluation context $B_{\{x,y\}}[\ ] = N[F_{\{x,y\}}[\ ]]$ that is given by Lemma 32 for the set $\{x, y\}$ with bound $k \in \mathbb{N}$, and some *int* $\notin S$. Let R_u be the interpreter given by Corollary 37, for processes of sort $S \cup \{x, y\}$, for some $u \notin S$. We build our context as follows:

$$\begin{aligned} T_u &\stackrel{\text{def}}{=} \overline{\mathit{int}}\langle u\rangle \oplus F_{\{x,y\}}[R_u] \\ G_n &\stackrel{\text{def}}{=} \nu c.(\overline{c}\langle n\rangle \mid\ !c(u).\overline{c}\langle u+1\rangle \mid\ c(u).T_u) \\ U_{S,n}[\ ] &\stackrel{\text{def}}{=} N\,[\nu S.(G_n \mid [\ ])] \\ U_S[\ ] &\stackrel{\text{def}}{=} U_{S,k}[\ ] \end{aligned}$$

If P has sort S, then R_u has sort $S \uplus \{x, y, u\}$, T_u has sort $S \uplus \{\mathit{int}, u\}$, $\nu S.(G_n \mid P)$ has sort $\{\mathit{int}\}$, and $U_{S,n}[P]$ has sort $\{x, y\}$. The process R_u interprets a π-calculus process, R, encoded by u. The process T_u either reveals the value u as an integer barb or silently reduces to $F_{\{x,y\}}[R_u]$. The process G_k chooses any integer $n \geq k$ as the result of an infinite internal choice $T_k \oplus (T_{k+1} \oplus (T_{k+2} \oplus \cdots))$: we have $G_n \approx_l T_n \oplus G_{n+1}$ for any $n \geq k$. Using these processes, the context $U_S[\ ]$ chooses an (integer-encoded) context $\nu S.(R \mid [\ ])$ encoded by n, then either reveals this choice of context or behaves like this context. We let the contexts $K_n[\ ]$ and $K'_n[\ ]$ be the derivatives of $U_{S,n}[\ ]$ at these intermediate stages, after choosing this particular n and after choosing to run the interpreter R_u with this $\langle\!\langle n \rangle\!\rangle_u$, respectively, and let G' and T' be the inert residues of these stages ($G' \sim_l \mathtt{0}$, $T' \sim_l \mathtt{0}$):

$$
\begin{aligned}
G' &\stackrel{\text{def}}{=} \nu c.!c(u).\bar{c}\langle u+1\rangle \\
T' &\stackrel{\text{def}}{=} \nu t.(t.\overline{int}\langle u\rangle \mid G') \\
K_n[\,] &\stackrel{\text{def}}{=} N\,[\nu S.\,(\nu u.(G' \mid \langle\!\langle n\rangle\!\rangle_u \mid T_u)\,|[\,])] \\
K'_n[\,] &\stackrel{\text{def}}{=} B\,[\nu S.\,(\nu u.(T' \mid \langle\!\langle n\rangle\!\rangle_u \mid R_u)\,|[\,])]
\end{aligned}
$$

We will use the following reduction property. Let Q be a process of sort S. If $U_S[Q] \to^* U'$, then there exists $n \geq k$ and Q' such that $Q \to^* Q'$ and one of the following holds:

(G) $U_S[Q] \to^* U_{S,n}[Q'] \equiv U'$.
(T) $U_S[Q] \to^* K_n[Q'] \equiv U'$.
(R) $U_S[Q] \to^* K'_n[Q'] \to^* U'$

The proof is by induction on the length of the derivation, and relies on Lemma 32(2). Crucially, only R_u shares names with the process placed in $U_{S,n}[\,]$. This process R_u is guarded until $K'_n[\,]$ appears in the reduction. Till then, reductions in the context and reductions from Q always commute.

Assume that P, Q, and $C[\,]$ meet the hypotheses of *(1)*. For all reductions $C[P] \to^* V$, we prove the existence of W such that $C[Q] \to^* W$ and $V \mathrel{\dot{\approx}_2} W$. There exists a process R of sort $S \cup \{x, y\}$ such that $C[P] \equiv \nu S.(P \mid R)$ and $C[Q] \equiv \nu S.(Q \mid R)$, with an integer encoding $[\![R]\!] \geq k$. Starting from $U_S[P]$, we build reductions representing $C[P] \to^* V$ with an interpreted $[\![R]\!]$, we use $\dot{\approx}_2$-bisimulations as given in the definition of ϕ_S, and we extract reductions $C[Q] \to^* W$. The situation is detailed in the diagram below, with the extracted reductions on the right.

(1) The upper square of the diagram deals with the internal choice of $n = [\![R]\!]$ in G_n. The top edge holds by definition of ϕ_S. On the left, we have reductions

$$U_S[P] \to U_{S,k+1}[P] \to \cdots \to U_{S,[\![R]\!]} \to K_{[\![R]\!]}[P]$$

By $\dot{\approx}_2$-bisimulation, we obtain reductions $U_S[Q] \to^* U'$ on the right, with $K_{[\![R]\!]}[P] \mathrel{\dot{\approx}_2} U'$. By construction, $K_{[\![R]\!]}[P] \to^* \mathrel{\dot{\approx}_2} N_{[\![R]\!]}$ and, by bisimulation, U' must have the same property. We show that the reductions $U_S[Q] \to^* U'$ are in case (T) of the reduction property, with $n = [\![R]\!]$. Otherwise:

(R) We have $K'_n[Q'] \mathrel{\dot{\approx}} B[\nu S, u.(\langle\!\langle n\rangle\!\rangle_u \mid R_u \mid Q')]$ and, by Lemma 32(3), $K'_n[Q'] \to^* \mathrel{\dot{\approx}_2} N_n$ only for $n < k$.

(T) with $n \neq [\![R]\!]$**.** We have reductions to (R), as discussed above, and to N_n up to $\dot{\approx}_2$, and thus $U' \to^* \mathrel{\dot{\approx}_2} N_n$ implies $n \neq [\![R]\!]$.

(G) We have $U' \to^* \mathrel{\dot{\approx}_2} N_n$ for some $n > [\![R]\!]$. However, the symmetric argument of the case above yields $K_{[\![R]\!]}[P] \to^* \mathrel{\dot{\approx}_2} N_m$ implies $m \neq n$.

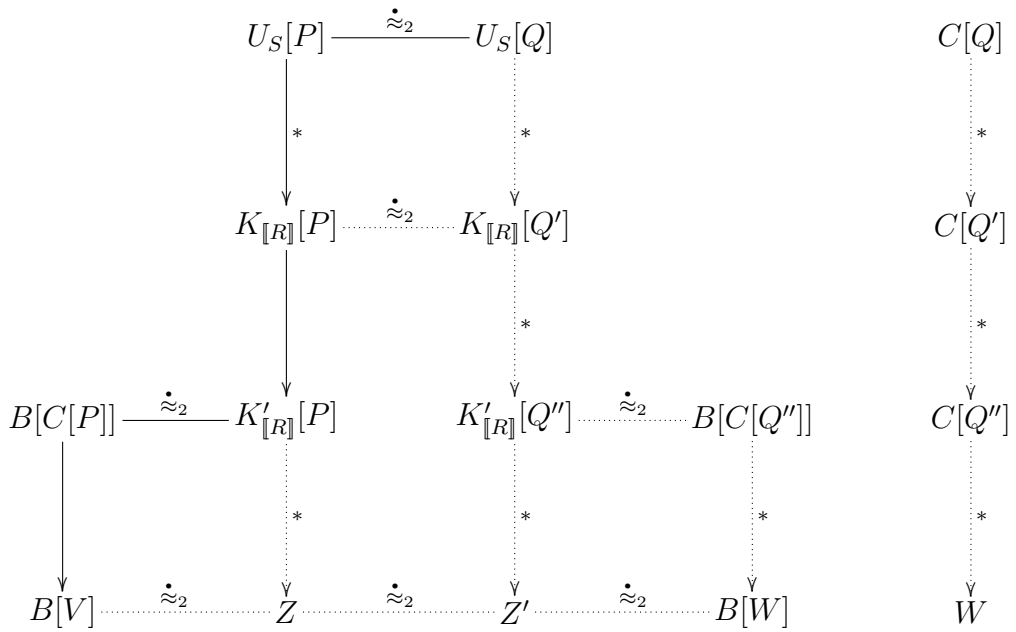
(2) Below, the reduction $K_{[\![R]\!]}[P] \to K'_{[\![R]\!]}[P]$ discards the integer barb $[\![R]\!]$ used as a marker for this particular $C[\,]$ and starts the interpreter. Using Corollary 37, the preservation of $\approx_l$ by application of the evaluation context $B[\,]$, and the inclusion $\approx_l \subseteq \dot{\approx}_2$, we obtain $K'_{[\![R]\!]}[P] \mathrel{\dot{\approx}_2} B[C[P]]$.

(3) In the bottom-left square, the reductions $C[P] \to^* V$ in context $B[\ ]$ are simulated by some $K'_{[\![R]\!]}[P] \to^* Z$ with $B[V] \mathrel{\dot{\approx}_2} Z$.

(4) In the central part of the diagram, the reductions $K_{[\![R]\!]}[P] \to^* K'_{[\![R]\!]}[P] \to^* Z$ can be simulated by $K_{[\![R]\!]}[Q'] \to^* Z'$ with $Z \mathrel{\dot{\approx}_2} Z'$.
Since $B[V] \mathrel{\dot{\approx}_2} Z'$ and, by Lemma 32(3), not $B[V] \to^* \mathrel{\dot{\approx}_2} N_{[\![R]\!]}$, the reductions $K_{[\![R]\!]}[Q'] \to^* Z'$ must be in case (R) for $n = [\![R]\!]$ and, for some Q'' with $Q' \to^* Q''$, we can split these reductions into $K_{[\![R]\!]}[Q'] \to^* K'_{[\![R]\!]}[Q''] \to^* Z'$. (However, there is no central $\mathrel{\dot{\approx}_2}$ edge and no obvious way to relate $C[P]$ and $C[Q'']$ at this stage.)

(5) In the bottom-right square, by Corollary 37, we obtain the top $\mathrel{\dot{\approx}_2}$ edge and, by simulation, $K'_{[\![R]\!]}[Q''] \to^* Z'$ implies $B[C[Q'']] \to^* Z''$ for some $Z'' \mathrel{\dot{\approx}_2} Z'$. Composing the $\mathrel{\dot{\approx}_2}$ equivalences at the bottom, we obtain $B[V] \mathrel{\dot{\approx}_2} Z''$. By Lemma 32(2), there exists W such that $C[Q''] \to^* W$ and $Z'' \equiv B[W]$.

(6) Composing the reductions on the right, we finally obtain $C[Q] \to^* W$ and $B[V] \mathrel{\dot{\approx}_2} B[W]$, that is, $V \mathrel{\dot{\approx}} W$ by Lemma 32(1).



We conclude the proof of property *(1)* of the lemma by showing that the relation $\{(C[P], C[Q]) \mid P\ \phi_S\ Q\} \cup \mathrel{\dot{\approx}_2}$ is a double-barbed bisimulation. We have just established a sufficient bisimulation property: if $C[P] \to^* V$, then $C[Q] \to^* W$ with $V \mathrel{\dot{\approx}_2} W$, and vice-versa. The preservation of the barbs $\Downarrow_x$ and $\Downarrow_y$ follows from the special case of an empty series of steps ($C[P] = V$): we obtain $C[Q] \to^* \mathrel{\dot{\approx}_2} C[P]$, hence $C[P] \Downarrow_x$ implies $C[Q] \Downarrow_x$ and $C[P] \Downarrow_y$ implies $C[Q] \Downarrow_y$.

The proof of property *(2)* of the lemma combines several instances of property *(1)*. In the definition of ϕ, we use the injective renamings to circumvent the limitation $\{x, y\} \notin S$. Assume $P\sigma\ \phi\ Q\sigma$ with $P\ \phi_S\ Q$.

Barbs: we let $C[\ ] = B_S[\ ]$, where the context $B_S[\ ]$ is given by Lemma 32, and obtain $B_S[P] \dot{\approx}_2 B_S[Q]$ by property *(1)*. By Lemma 32(1), we have $P \dot{\approx} Q$. In particular P and Q have the same weak barbs, and $P\sigma$ and $Q\sigma$ have the same barbs.

Bisimulation: if $P \to^* P'$, by definition of ϕ_S, we have $U_S[P] \dot{\approx}_2 U_S[Q]$, and the reductions $U_S[P] \to^* U_S[P']$ is simulated by some $U_S[Q] \to^* U'$. Both series of reductions are in case (G) for $n = k$, since otherwise we don't have $U' \to^* \dot{\approx}_2 N_n$ for all $n \geq k$ (Lemma 32(3)). We obtain $Q \to^* Q'$ with $U' \equiv U_S[Q']$, and finally $P' \ \phi_S \ Q'$. Finally, for all injective renamings σ and processes P, we have $P\sigma \to P'\sigma$ if and only if $P \to P'$.

Context closure: for a given evaluation context $C''[\ ]$, there exist an evaluation context $C'[\ ]$ and an injective renaming σ' such that $x, y \notin \mathrm{fv}(C'[\ ])$, $C''[P\sigma] = (C'[P])\sigma'$, and $C''[Q\sigma] = (C'[Q])\sigma'$. (If x appears in the sort of $C''[\ ]$, we pick a fresh name x' and let $\sigma' = \sigma\{^x/_{x'}\}$, and similarly for y.)
We let $S' = S \cup \mathrm{fv}(C'[\ ])$ and $C[\ ] = U_{S'}[C'[\ ]]$. By property *(1)* for S and $C[\ ]$, we obtain $U_{S'}[C'[P]] \dot{\approx}_2 U'_S[C'[Q]]$, which is the definition of $C'[P] \ \phi_{S'} \ C'[Q]$, and thus $C''[P\sigma] \ \phi \ C''[Q\sigma]$. □

Proof of Theorem 1 To conclude, we prove the inclusion $\dot{\approx}^\circ_2 \subseteq \approx$. Assume $P \dot{\approx}^\circ_2 Q$ and let $S = \mathrm{fv}(P) \cup \mathrm{fv}(Q)$. If $x, y \notin S$, by congruence property, we have $U_S[P] \dot{\approx}_2 U_S[Q]$. By Lemma 39(2), we obtain $P \approx Q$. If x or y appear in S, we similarly obtain $P\sigma \approx Q\sigma$ for some injective renaming σ, hence $P \approx Q$. □

8 Labels instead of Barbs and Contexts

Bisimulation proofs of barbed congruences still require some explicit context closure, as for instance in most proofs of [15,3]. This is not the case for labeled bisimulations, where congruence is a derived property instead of a requirement in the definition of equivalence. Thus, purely co-inductive proof techniques suffice to establish equivalences. We write $\approx_l$ for (weak) labeled bisimilarity, and refer to [43,6,17] for various formulations of $\approx_l$ for asynchronous process calculi and their impact on proof techniques.

Considered as an auxiliary semantics for a reduction system, a labeled transition system is *sound* at least when its silent τ-transitions coincide with the reductions ($\xrightarrow{\tau}{}^* = \to^*$) and when its labeled transitions determine the observation predicates $\Downarrow_x$. Then, any (weak) labeled bisimulation is also a barbed bisimulation. Considered as a proof technique for observational equivalences, labeled bisimulation is sound ($\approx_l \subseteq \approx$) when, in addition, labeled bisimilarity is closed by application of all contexts used in the definition of $\approx$. This is usually the case, inasmuch as labels are meant to represent elementary contexts.

In the π-calculus, we have $P \downarrow_x$ if and only if $P \xrightarrow{\alpha}$ for some output label of the form $\alpha = (\widetilde{z})\overline{x}\langle\widetilde{y}\rangle$; $\approx_l$ is closed by restriction and parallel contexts (see [6]); hence we have the well-known inclusions $\approx_l \subseteq \approx$ and $\approx_l \subseteq \dot{\approx}^\circ$. The first inclusion is strict because our evaluation contexts have less discriminating power than labels. For instance, the key barbed congruence for equators, recalled in Proposition 8, is not a labeled bisimulation. Whereas the process E_x^y silently converts messages between x and y, one can still distinguish x from y as an argument in output transitions. For instance, $E_x^y \mid \overline{z}\langle x\rangle \not\approx_l E_x^y \mid \overline{z}\langle y\rangle$ because the labels $\overline{z}\langle x\rangle$ and $\overline{z}\langle y\rangle$ are not equated.

To fix this discrepancy, the usual approach is to extend the syntax with a name matching prefix, such as $[x = y]P$. In the extended calculus, each label can then be tested by a specific context through a series of name matchings, and thus barbed congruence should coincide with some variant of labeled bisimulation. (Although labeled bisimulations may be easier to establish, name matching is a mixed blessing. It is usually not a primitive in higher-order settings. Technically, it also induces additional subtleties [40], and breaks properties such as the stability of equivalence by substitution. Many useful equations that are proper to asynchronous calculi disappear [27]. Besides, labeled bisimulations may be too fine even in presence of name matching in the syntax [1].)

In the π-calculus with name matching, early bisimulation and barbed congruence coincide, but the proof is delicate—this is mentioned as an open question in [32]. To our knowledge, the only general statement of their coincidence appears in Sangiorgi's thesis [39], with a proof of the problematic inclusion $\dot{\approx}^\circ \subseteq \approx_l$ for both CCS and the monadic π-calculus; the technique consists of building contexts that test for all possible behaviors of a process under bisimulation, and that exhibit different barbs accordingly. This technique requires infinite contexts with infinitely-many recursive constants and free names. These extended contexts are never considered in the usual congruence properties for the π-calculus, and they cannot be expressed using the simpler constructs of asynchronous calculi.

In other works, partial results are obtained for variants of the calculus (CCS [32], the asynchronous π-calculus [6]). The proof techniques are similar, but use only finite contexts. As a result, the coincidence is established only for *image finite* processes. A process P is image finite when the set of its derivatives is finite. In the case of weak relations, this means in particular that $\{P', P \rightarrow^* P'\}$ has to be finite. This restriction is annoying, especially as many processes that use replication (or even replicated input) are not image-finite by series of reductions. For instance, we have $!(\tau.P) \rightarrow^* !(\tau.P) \mid P \mid \ldots \mid P$ and similarly $Q = \overline{x} \mid !x.(P \mid \overline{x}) \rightarrow^* Q \mid P \mid \ldots \mid P$.

Theorem 5 *In the π-calculus with name matching, we have $\dot{\approx}^\circ = \approx_l$.*

We could adapt Sangiorgi's proof by replacing all free names by integers, as Lemmas 31 and 34 would provide a finite encoding of his infinite contexts. Actually,

there is a much simpler proof at hand: we prove the inclusion $\approx \subseteq \approx_l$ then apply Theorem 1. A proof of the inclusion $\approx \subseteq \approx_l$ already appears in [23], in a similar setting. Our proof, however, is significantly shorter, and illustrates the advantage of the congruence-and-bisimulation definition of equivalence. Instead of capturing the whole synchronization tree in a huge context, we exhibit for every labeled transition a context that specifically detects this transition, then disappears up to barbed congruence. The proof relies on the following technical lemma:

Lemma 40 (accommodating the extrusions) *In the π-calculus, with or without name matching, let P, Q be processes and $y \notin \mathrm{fv}(P, Q)$. We have $P \approx Q$ if and only if $\nu x.(\overline{y}\langle x\rangle \mid P) \approx \nu x.(\overline{y}\langle x\rangle \mid Q)$.*

Intuitively, the evaluation contexts $E_{x,y}[\ ] \stackrel{\text{def}}{=} \nu x.(\overline{y}\langle x\rangle \mid [\ ])$ represent the residues of contexts that test for output labels of the form $(x)\overline{z}\langle\widetilde{w}\rangle$ that extrude x.

PROOF. Since $\approx$ is closed by application of evaluation contexts, if $P \approx Q$, then also $E_{x,y}[P] \approx E_{x,y}[Q]$. Conversely, let $\mathcal{R}$ be the relation that contains all pairs of processes (P, Q) such that, for some y not free in P, Q, we have $E_{x,y}[P] \approx E_{x,y}[Q]$. We show that $\mathcal{R}$ is a congruence and a barbed bisimulation.

(Strong) bisimulation: reduction steps in P and $E_{x,y}[P]$ are in direct correspondence: if $P \to P'$, then $E_{x,y}[P] \to E_{x,y}[P']$, and conversely if $E_{x,y}[P] \to T$, then we can exhibit some process P' such that $P \to P'$ and $T \equiv E_{x,y}[P']$. (Since $y \notin \mathrm{fv}(P)$, the message $\overline{y}\langle x\rangle$ remains inert.)

(Strong) barbs: assume $y, t \notin \mathrm{fv}(P)$. We never have $P \downarrow_y$. We have $P \Downarrow_x$ if and only if $E_{x,y}[P] \mid y(x).x(\widetilde{u}).\bar{t} \Downarrow_t$. For any $z \notin \{x, y\}$, we have $P \downarrow_z$ if and only if $E_{x,y}[P] \downarrow_z$.

Context closure: without loss of generality, we consider only contexts of the form $C[\ ] \stackrel{\text{def}}{=} \nu\widetilde{v}.(R \mid [\ ])$ and we exhibit a context $C'[\ ]$ such that for any processes P with $y, z \notin \mathrm{fv}(P) \cup \{\widetilde{v}\}$, we can commute contexts up to equivalence: $C'[E_{x,y}[P]] \approx E_{x,z}[C[P]]$. When $C[\ ]$ does not restrict x ($x \notin \widetilde{v}$), we use:

$$C'[\ ] \stackrel{\text{def}}{=} \nu y.\nu\widetilde{v}.([\ ] \mid y(x).(\overline{z}\langle x\rangle \mid R))$$

Otherwise, we use the context

$$C'[\ ] \stackrel{\text{def}}{=} E_{x,z}[\mathsf{0}] \mid \nu y.\nu\widetilde{v}.([\ ] \mid y(x).R)$$

(In both cases, we actually prove a finer, labeled bisimulation; we omit these proofs.) To conclude, suppose $P \mathcal{R} Q$, that is, $E_{x,y}[P] \approx E_{x,y}[Q]$. Since $\approx$ is a congruence, we have $C'[E_{x,y}[P]] \approx C'[E_{x,y}[Q]]$. By transitivity, we obtain $E_{x,z}[C[P]] \approx E_{x,z}[C[Q]]$, that is, $C[P] \mathcal{R} C[Q]$. □

In combination with our previous results, this establishes the coincidence of labeled bisimulation and barbed congruence in the presence of name matching:

Theorem 6 *In the π-calculus with name matching, we have $\approx = \approx_l$.*

PROOF. We prove $\approx \subseteq \approx_l$ by establishing that $\approx$ is a labeled bisimulation. Let $P \approx Q$ and $P \stackrel{\alpha}{\to} P'$. We build a specific context for every label α.

Internal step: in case $P \to P'$, the bisimulation requirement of $\approx$ suffices to obtain $Q \to^* Q'$ with $P' \approx Q'$.

Input action: in case $P \stackrel{x(\tilde{y})}{\longrightarrow} P'$, we have $P' \equiv P \,|\, \overline{x}\langle\tilde{y}\rangle$. We always have $Q \stackrel{x(\tilde{y})}{\longrightarrow} Q' \stackrel{\text{def}}{=} Q \,|\, \overline{x}\langle\tilde{y}\rangle$. We use congruence for the context $[\,] \,|\, \overline{x}\langle\tilde{y}\rangle$ to obtain $P' \approx Q'$.

Output action: we only consider the case $P \stackrel{(z)\overline{x}\langle y,z\rangle}{\longrightarrow} P'$ where the process P outputs a single free name y and a single bound name z (being extruded). The general case easily follows. We apply the congruence property for the context

$$T[\,] \stackrel{\text{def}}{=} \bar{t} \,|\, x(y',z).(\,\overline{u}\langle z\rangle \,|\, [y=y']t \,|\, \textstyle\prod_{y\in \mathrm{fv}(P)}[y=z]\bar{t}\,)$$

where t is a name that does not occur in P or Q. The message $\bar{t}$ is used as a barb that disappears only if the process in $T[\,]$ produces an output with label $\stackrel{(z)\overline{x}\langle y,z\rangle}{\longrightarrow}$. That is, $T[P] \to\to\to E_{z,u}[P']$ and, whenever $T[Q] \to^* T'$ with $T' \not\Downarrow_t$, there is a process Q' such that $Q \to^* \stackrel{(z)\overline{x}\langle y,z\rangle}{\longrightarrow} \to^* Q'$ and $T' \equiv E_{z,u}[Q']$. Then, $T[P] \approx T[Q]$ and $T[P] \to^* E_{z,u}[P']$ yields by bisimulation such a Q' with $T[Q] \to^* E_{z,u}[Q']$ (since $E_{z,u}[P'] \not\Downarrow_t$) and $E_{z,u}[P'] \approx E_{z,u}[Q']$. We conclude by Lemma 40. □

9 A Family Portrait (Summary)

We finally gather our results in a hierarchy of equivalences. Figure 3 deals with the general case of a reduction system equipped with a notion of evaluation context, and compares the main congruences considered in this paper. All solid lines represent inclusions between relations (which may or may not be strict). These inclusions directly follow from the definitions. The same inclusions hold for any choice of derived observation predicates: committed, existential, or committed-existential. In practice, for process calculi, we expect the additional inclusion $\simeq_{\text{fair}} \subseteq \simeq_{\text{may}}$, and also that at least the tiers with dotted horizontal lines remain different: $\approx \subset \lessapprox \subset \simeq_{\text{fair}} \subset \simeq_{\text{may}}$.

Figure 4 deals with our asynchronous π-calculus, in the absence of name matching. It combines results obtained for different variants of our equivalences, for different choices of observation predicates, as discussed in Sections 5 and 6. We omit the existential variants for the congruences $\approx$, $\lessapprox$, $\simeq_{\text{fair}}$, and $\simeq_{\text{may}}$—they all coincide with their base equivalence. With name matching, the two upper tier also coincide.

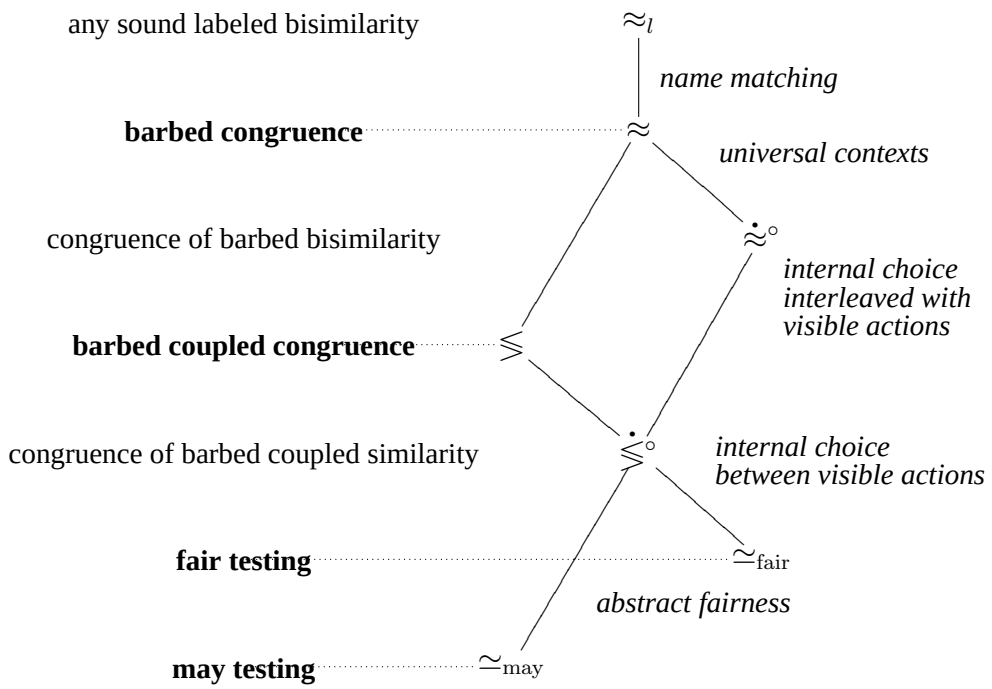


Fig. 3. General inclusions between asynchronous congruences

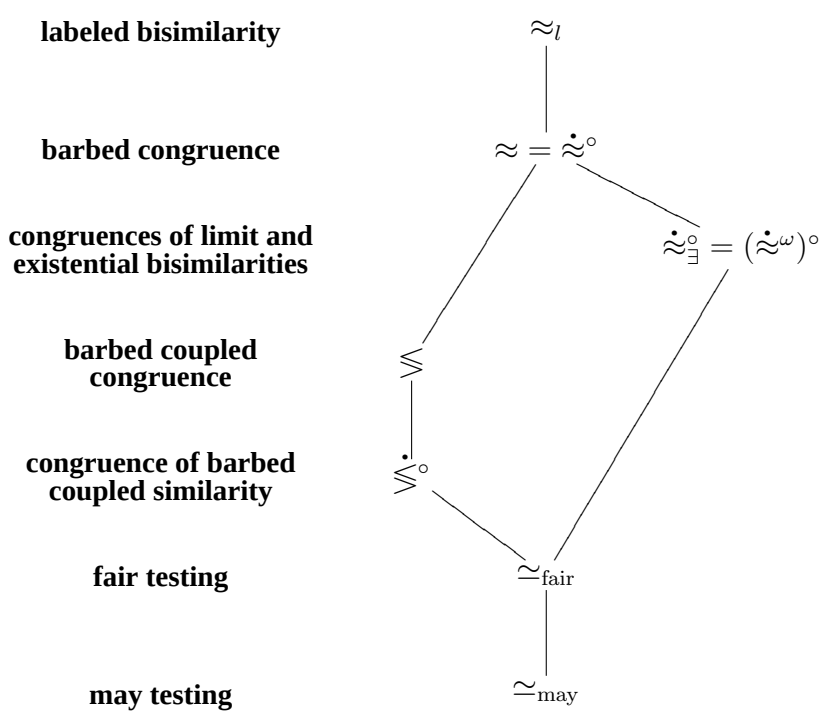


Fig. 4. Strict inclusions in the asynchronous π-calculus

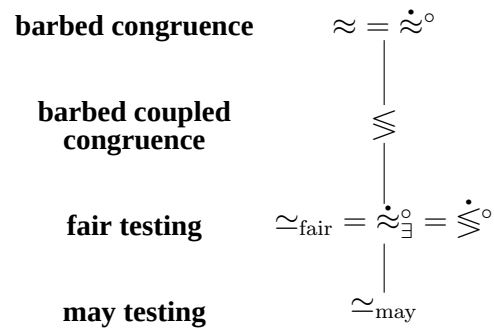


Fig. 5. Strict inclusions in the local π-calculus

Figure 5 deals with the simpler hierarchy obtained for the local π-calculus, with the same conventions.

Almost all interesting results seem specific to the π-calculus, inasmuch as their proofs rely on specific contexts and encodings. However, we believe that the basic techniques can be carried over to many variants of the π-calculus and to similar process calculi. This is certainly the case for the join calculus; despite the significant differences discussed in Section 6, and a few twists in the main proofs [17,14,13]. Some techniques have also been usefully applied to Cardelli and Gordon's calculus of Mobile Ambients [19], and to mobile process calculi with cryptographic primitives [3,2,1].

References

[1] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL 2001: Proceedings 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 104–115. ACM, January 2001.

[2] Martín Abadi, Cédric Fournet, and Georges Gonthier. Authentication primitives and their compilation. In *27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2000)*, pages 302–315. ACM, January 2000.

[3] Martín Abadi, Cédric Fournet, and Georges Gonthier. Secure implementation of channel abstractions. *Information and Computation*, 174(1):37–83, April 2002.

[4] Martín Abadi and Andrew D. Gordon. Reasoning about cryptographic protocols in the spi calculus. In Mazurkiewicz and Winkowski [26], pages 59–73.

[5] Gul Agha, Ian Mason, Scott Smith, and Carolyn L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7(1):1–72, January 1997.

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

[7] Michele Boreale and Rocco De Nicola. Testing equivalence for mobile processes. *Information and Computation*, 120(2):279–303, August 1995.

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

[9] Ed Brinksma, Arend Rensink, and Walter Vogler. Fair testing. In I. Lee and S. A. Smolka, editors, *6th International Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *Lecture Notes in Computer Science*, pages 313–327. Springer-Verlag, 1995.

[10] Ed Brinksma, Arend Rensink, and Walter Vogler. Applications of fair testing. In R. Gotzhein and J. Bredereke, editors, *Formal Description Techniques IX: Theory, Applications and Tools*, volume IX. Chapman and Hall, 1996.

[11] R. Cleaveland, editor. *Third International Conference on Concurrency Theory (CONCUR'92)*, volume 630 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[12] Rocco De Nicola and Matthew C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[13] C. Fournet and G. Gonthier. The join calculus: a language for distributed mobile programming. In G. Barthe, P .Dybjer, , L. Pinto, and J. Saraiva, editors, *Proceedings of the Applied Semantics Summer School (APPSEM), Caminha, September 2000*, volume 2395 of *Lecture Notes in Computer Science*, pages 268–332. Springer-Verlag, August 2002.

[14] Cédric Fournet. *The Join-Calculus: a Calculus for Distributed Mobile Programming*. PhD thesis, Ecole Polytechnique, Palaiseau, November 1998.

[15] Cédric Fournet and Georges Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Conference record of the 23th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'96)*, pages 372–385. ACM, January 1996.

[16] Cédric Fournet and Georges Gonthier. A hierarchy of equivalences for asynchronous calculi (extended abstract). In Larsen et al. [25], pages 844–855.

[17] Cédric Fournet and Cosimo Laneve. Bisimulations in the join-calculus. *Theoretical Computer Science*, 266(1-2):569–603, September 2001.

[18] Cédric Fournet, Cosimo Laneve, Luc Maranget, and Didier Rémy. Implicit typing à la ML for the join-calculus. In Mazurkiewicz and Winkowski [26], pages 196–212.

[19] Cédric Fournet, Jean-Jacques Lévy, and Alan Schmitt. An asynchronous, distributed implementation of mobile ambients. In J. van Leeuwen, O. Watanabe, M. Hagiya, P.D. Mosses, and T. Ito, editors, *Proceedings of IFIP TCS 2000*, volume 1872 of *Lecture Notes in Computer Science*. IFIP TC1, Springer-Verlag, August 2000.

[20] Rob J. van Glabbeek. The linear time—branching time spectrum II; the semantics of sequential systems with silent moves (extended abstract). In E. Best, editor, *4th International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, 1993.

[21] Matthew Hennessy. *Algebraic Theory of Processes*. The MIT Press, 1988.

[22] Kohei Honda and Mario Tokoro. On asynchronous communication semantics. In P. Wegner, M. Tokoro, and O. Nierstrasz, editors, *Proceedings of the ECOOP'91 Workshop on Object-Based Concurrent Computing*, volume 612 of *Lecture Notes in Computer Science*, pages 21–51. Springer-Verlag, 1992.

[23] Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[24] Cosimo Laneve. May and must testing in the join-calculus. Technical Report UBLCS 96-04, University of Bologna, March 1996. Revised: May 1996.

[25] Kim Larsen, Sven Skyum, and Glynn Winskel, editors. *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP '98)*, volume 1443 of *Lecture Notes in Computer Science*. Springer-Verlag, July 1998.

[26] A. Mazurkiewicz and J. Winkowski, editors. *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *Lecture Notes in Computer Science*. Springer-Verlag, July 1997.

[27] Massimo Merro and Davide Sangiorgi. On asynchrony in name-passing calculi. In Larsen et al. [25], pages 856–867.

[28] Robin Milner. *A Calculus of Communicating Systems*, volume 92. Springer-Verlag, 1980. Lecture Notes in Computer Science.

[29] Robin Milner. *Communication and Concurrency*. Prentice Hall, New York, 1989.

[30] Robin Milner. The polyadic π-calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*. Springer-Verlag, 1993.

[31] Robin Milner. *Communication and Mobile Systems: the π-Calculus*. Cambridge University Press, Cambridge, 1999.

[32] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings of ICALP'92*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer-Verlag, 1992.

[33] James H. Morris, Jr. *Lambda-Calculus Models of Programming Languages*. Ph. D. dissertation, MIT, December 1968. Report No. MAC–TR–57.

[34] V. Natarajan and Rance Cleaveland. Divergence and fair testing. In *Proceedings of ICALP '95*, volume 944 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[35] Uwe Nestmann and Benjamin C. Pierce. Decoding choice encodings. *Information and Computation*, 163:1–59, Nov 2000.

[36] D. M. R. Park. *Concurrency and Automata on Infinite Sequences*, volume 104 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[37] Joachim Parrow and Peter Sjödin. Multiway synchronization verified with coupled simulation. In Cleaveland [11], pages 518–533.

[38] Joachim Parrow and Peter Sjödin. The complete axiomatization of cs-congruence. In P. Enjalbert, E. W. Mayr, and K. W. Wagner, editors, *Proceedings of STACS'94*, volume 775 of *Lecture Notes in Computer Science*, pages 557–568. Springer-Verlag, 1994.

[39] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. Ph.D. thesis, Department of Computer Science, University of Edinburgh, 1992.

[40] Davide Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33:69–97, 1996.

[41] Davide Sangiorgi. On the bisimulation proof method. *Journal of Mathematical Structures in Computer Science*, 8:447–479, 1998.

[42] Davide Sangiorgi and Robin Milner. The problem of "weak bisimulation up to". In Cleaveland [11], pages 32–46.

[43] Davide Sangiorgi and David Walker. *The Pi-calculus: a Theory of Mobile Processes*. Cambridge University Press, July 2001.