Bisimulation in Name-Passing Calculi without Matching*

Michele Boreale [†]

Università "La Sapienza"
Rome, ITALY

Davide Sangiorgi [‡]

INRIA
Sophia Antipolis, FRANCE

Abstract

We study barbed equivalence in name-passing languages where there is no matching construct for testing equality between names. We concentrate on the π-calculus with capability types and subtypes, of which the untyped π-calculus without matching is a special case. We give a coinductive characterisation of typed barbed equivalence, and present "bisimulation up–to" techniques to enhance the resulting coinductive proof method. We then use these techniques to prove some process equalities that fail in the ordinary π-calculus.

1 Introduction

The π-calculus is a development of CCS where names may be communicated. Name-passing permits the modeling of systems with dynamic linkage reconfigurations. The π-calculus is being used for giving the semantics to higher-order, object-oriented, constraint, concurrent languages [24, 20, 23], and as a basis for the design of new programming languages [17, 3]. The theory of the π-calculus is well-developed. At its heart are the definitions of behavioural equivalence, which are needed for proving, for instance, program transformations. Behavioural equivalence means "undistinguishability in all contexts". Typically a notion of success (or convergence) is defined, and then two terms are declared behaviourally equivalent if they have the same success properties *in all contexts. Barbed equivalence* is defined in this way, and is one of the most studied behavioural equivalences for the π-calculus.

Context-based behavioural equalities like barbed equivalence suffer from the universal quantification on contexts, that makes it very hard to prove process equalities following the definition, and makes mechanical checking impossible. Against this problem, it is important to find *direct characterisations*, without quantification on contexts. In the case of barbed equivalence such a characterisation is given by *labeled bisimilarity* (in the *early* style). Approximately, P and Q are bisimilar if

$$P \xrightarrow{\mu} P' \quad \text{implies } Q \overset{\mu}{\Longrightarrow} Q', \quad \text{for some } Q' \text{ bisimilar to } P' \tag{1}$$

and the vice versa, on the possible transitions by Q. This is a more useful definition for proving process equalities, because it is coinductive and has no quantification on contexts. The coinduction proof method can be enhanced with "up to techniques", such as bisimulation up to bisimulation, up to restriction, up to context [11, 19]. Moreover, coinductive characterisations like (1) can be the basis of tools for mechanically checking process equalities [22].

When proving characterisations of barbed equivalence in terms of labeled bisimulations like (1), a central role is played by the *matching* construct (or similar constructs, like mismatching), that is used for testing equality between names. Indeed, bisimulation (1) implicitly gives the observer the capability of looking at the names receive by the process, because labels of matching transitions must be syntactically the same. This prevents us from proving useful program transformations (something similar happens in imperative languages where it is possible to test equality between pointers). Another argument against the matching construct is that matching on names (precisely, on names that are actually used as channels) is rarely used in programming. For instance, there is no matching on names in Pict and Join. Mostly

†Address: Dipartimento di Scienze dell'Informazione, V. Salaria 113 - III piano, 00198 Roma Italy. Email: `michele@dsi.uniroma1.it`

‡Address: INRIA-Sophia Antipolis, B.P. 93 F-06902 Sophia Antipolis France. Email: `davide.sangiorgi@sophia.inria.fr`.

important, matching is often explicitely forbidden in *typed* π-calculi; for instance in presence of *capability types* (sometimes called I/O types) [15]. These types, similar to Reynold's reference types for Forsythe [18], allow us to distinguish, for instance the capability of using a name in input from that of using the name in output. If a name is received with only the input or only the output capability, it cannot be tested for equality (which would mean going beyond the allowed capability). Capabilities are useful for protecting resources; for instance, in a client-server model, they can be used for preventing clients from using the access name to the server in input and stealing messages to the server; similarly they can be used in distributed programming for expressing security constraints [4]. Capabilities are ubiquitous in the Join and Pict languages: in Join, only the output capabilities on names may be communicated; in Pict it is rare or forbidden that a name is passed with all capabilities. Capabilities give rise to *subtyping*: the output capability is contravariant, whereas the input capability is covariant. Subtyping is useful when π-calculus is used for object-oriented programming, or for giving the semantics to object-oriented languages.

In this paper, we study barbed equivalence in π-calculus languages without matching. We concentrate on the π-calculus with capability types, of which the untyped π-calculus without matching is a special case. Our main result is a coinductive characterisation of typed barbed equivalence. We also present several "bisimulation up–to" techniques to enhance the resulting coinductive proof method. We then apply these techniques to show behavioural equalities between processes that fail in the ordinary π-calculus. In the remainder of this introduction, we discuss capability types, their semantic consequences, and give and an overview of the technical developments in the paper.

In the recent proposals of *types* and *subtypes* for the π-calculus, types are assigned to names, and impose a discipline on what names can carry. A capability type shows the capability of a name and, recursively, of the names carried by that name. For instance, a type[1] $p : T^{\mathsf{o}\mathsf{i}}$ (for appropriate type expression T) says that name p can be used only in input; moreover, any name received at p may only be used in output. Thus, process $p(q).\,\overline{q}r.\,\mathbf{0}$ is well-typed under the type assignment $p : T^{\mathsf{o}\mathsf{i}}, r : T$. (We recall that $\overline{q}r.\,P$ is the output at q of name r with continuation P, and that $p(q).\,P$ is an input at p with q placeholder for the name received in the input. Below we shall also use the prefix $\overline{a}(b).\,P$, that represents the output at a of a private name b; parallel composition $P \mid Q$; and replication $!P$, that represents an infinite number of copies of P in parallel. We write $\overline{p}.P$ and $p.\,P$ when the name transmitted at p is not important, and we often abbreviate a prefix $\pi.\mathbf{0}$ as π). To see why the addition of capability types has semantic consequences, we discuss some examples. Let

$$\begin{array}{rcl} P & \stackrel{\text{def}}{=} & \overline{b}(x).\,a(y).\,(\overline{y} \mid x) \\ Q & \stackrel{\text{def}}{=} & \overline{b}(x).\,a(y).\,(\overline{y}.x + x.\overline{y}) \end{array} \qquad (2)$$

These processes are not behaviourally equivalent in the untyped π-calculus. For instance, they are not early bisimilar because P may terminate after 2 communications with the external observer, where the transitions performed by P are $P \xrightarrow{\overline{b}(x)} \xrightarrow{ax} \xrightarrow{\tau} \mathbf{0}$. By contrast, Q always terminate after 4 interactions with the observer. However, if we impose that only the input capability of names may be communicated at b, then P and Q cannot be distinguished by any (well-typed) context. Similarly, processes

$$\begin{array}{rcl} P & \stackrel{\text{def}}{=} & \overline{a}(x).\,\overline{a}(y).\,(!y.R \mid !x.R) \\ Q & \stackrel{\text{def}}{=} & \overline{a}(x).\,\overline{a}x.\,(!x.R) \end{array} \qquad (3)$$

are not equivalent in the untyped π-calculus. They are not bisimilar because P may perform two initial outputs of private names, whereas Q emits twice the same name. However, P and Q become undistinguishable if only the output capability on names may be communicated at a.

Other examples may be found in [15], or in Section 7. Examples like (2) and (3) show that the theory of the untyped π-calculus is unsatisfactory in the calculus with capabilities (and more generally in π-calculus languages without matching). They also show that if we look for a *typed bisimilarity*, to replace the untyped definition (1), then at least two new factors have to be taken into account:

(a) Only a (possibly proper) subset of the actions of processes is observable: processes may be tested by an external observer only on those actions for which the observer has the necessary capabilities. For instance, in Example (2) the observer receives the input capability on x and therefore cannot resend it at a because this would require possessing the output capability.

(b) The labels of matching transitions of bisimilar processes may be syntactically different. For instance, in Example (3), P has a sequence of tran-

[1]Expression $T^{\mathsf{o}\mathsf{i}}$ is the same as $(T^{\mathsf{o}})^{\mathsf{i}}$.

sitions $P \xrightarrow{\overline{a}(x)} \xrightarrow{\overline{a}(y)} \xrightarrow{yv}$ that is matched by the sequence of transitions $Q \xrightarrow{\overline{a}(x)} \xrightarrow{\overline{a}x} \xrightarrow{xv}$ from Q.

To accommodate (a) and (b), we define bisimilarity of processes relative to a *typed closure*. The closure records the capabilities on names that an external observer may have acquired, and separates the observer's view on the identity of names from their real identity. Closures give us information on the names available at some point of the life of the external observer. This is similar to their use in the semantics of imperative programs, where closures show the type and value of the variables that are available at some point of the execution of a program. A closure $[\Delta, \sigma]$ is composed of: a type environment Δ, that collects the names known by the observer (one may think of these as variables local to the observer), together with the types that the observer has on them; a function σ from names to names, that says what is the real value of names in Δ. A closure $[\Delta, \sigma]$ together with a process P form a *configuration*, written $[\Delta,\ \sigma]\,\#\,P$.

We define transitions on configurations. They may be of three forms. The first form is

$$[\Delta,\ \sigma]\,\#\,P \xrightarrow[\delta]{\alpha} [\Delta',\ \sigma']\,\#\,P'.$$

This represents an interaction between the process and the observer in which α is the action performed by the process and $\delta\sigma'$ the action performed by the observer. δ is the observer's point of view on the action he performs (we call this a *virtual* action); the use of names in δ must respect the types assigned to them in Δ. For instance, if $P \stackrel{\mathrm{def}}{=} a(c).\,P'$, $\Delta \stackrel{\mathrm{def}}{=} x : T^{\mathsf{o}}, y : T$, and $\sigma \stackrel{\mathrm{def}}{=} \{x \to a, y \to b\}$, then we have

$$[\Delta,\ \sigma]\,\#\,P \xrightarrow[\overline{x}y]{ab} [\Delta,\ \sigma]\,\#\,P'\{c \to b\}$$

The closure grows if the observer creates a new name and passes it to the process, or if the process performs an output and the observer an input, like in

$$[\Delta,\ \sigma]\,\#\,P \xrightarrow[xz]{\overline{a}b} [\Delta_{[z:T]},\ \sigma_{[z\to b]}]\,\#\,P'$$

Here z is a fresh name (that is, it does not occur in Δ, σ and P), and $\Delta_{[z:T]}$, $\sigma_{[z\to b]}$ indicate the updates of Δ and σ on z. The fresh name z is introduced because the observer cannot look at the identity of the received name; in future actions, the observer will use z to refer to the name received. Note that two names are mapped onto the same name b in $\sigma_{[z\to b]}$ (*aliasing*).

The second form of transition for a configuration is

$$[\Delta,\ \sigma]\,\#\,P \xrightarrow[-]{\tau} [\Delta,\ \sigma]\,\#\,P',$$

and represents an interaction that takes place in the process, without the participation of the observer. The third form is

$$[\Delta,\ \sigma]\,\#\,P \xrightarrow[\langle\mu,\,\lambda\rangle]{-} [\Delta',\ \sigma']\,\#\,P$$

and represents an interaction that takes place in the observer, without the participation of the process. Actions μ and λ are virtual actions of the observer whose corresponding real actions are the dual of each other and therefore can be combined into an interaction. For instance, if $\Delta \stackrel{\mathrm{def}}{=} x : T^{\mathsf{i}}, y : T^{\mathsf{o}}, z : T$, and $\sigma \stackrel{\mathrm{def}}{=} \{x \to b, y \to b, z \to c\}$ then we have

$$[\Delta,\ \sigma]\,\#\,P \xrightarrow[\langle\overline{y}z,\,x(u)\rangle]{-} [\Delta_{[u:T]},\ \sigma_{[u\to c]}]\,\#\,P.$$

In our typed bisimilarity, we only compare configurations $[\Delta,\ \sigma]\,\#\,P$ and $[\Delta,\ \rho]\,\#\,Q$ with the same observer Δ. Suppose $[\Delta,\ \sigma]\,\#\,P$ takes a step

$$[\Delta,\ \sigma]\,\#\,P \xrightarrow[\delta]{\alpha} [\Delta',\ \sigma']\,\#\,P'.$$

We require that a matching transition from $[\Delta,\ \rho]\,\#\,Q$ be of the form

$$[\Delta,\ \rho]\,\#\,Q \xRightarrow[\delta]{\beta} [\Delta,\ \rho]\,\#\,Q'.$$

The observer's virtual actions (δ) are the same, but the real actions (the duals of α and β) may be different.

On the other hand, we allow that a transition

$$[\Delta,\ \sigma]\,\#\,P \xrightarrow[\langle\mu,\,\lambda\rangle]{-} [\Delta',\ \sigma']\,\#\,P$$

be matched by a sequence of transitions of the form

$$[\Delta,\ \rho]\,\#\,Q \xRightarrow[\mu]{\alpha} \xRightarrow[\lambda]{\beta} [\Delta',\ \rho']\,\#\,Q'.$$

In the transition from $[\Delta,\ \sigma]\,\#\,P$, only the observer moves. By contrast, in the transition from $[\Delta,\ \rho]\,\#\,Q$, observer and process cooperate to reach a configuration that is bisimilar with $[\Delta',\ \sigma']\,\#\,P$.

Since our bisimulation equates processes with syntactically different transitions, some basic proofs, like congruence for parallel composition, are rather different from the untyped case. In this short version of the paper most of the proofs are omitted.

2 Typed π-calculus

We present the typed π-calculus, following [15]. The syntax is given in Figure 1. The process constructs are those of the monadic π-calculus [13] but without matching. For simplicity of presentation, we chose a monadic, rather than polyadic, calculus, we do not have recursive types, and restrictions are not explicitly typed (we are not interested in type inference or type checking in this paper). We use σ to range over substitutions; for any expression E, we write $E\sigma$ for the result of applying σ to E, with the usual renaming convention to avoid captures. We assign sum and parallel composition the lowest precedence among the operators. The labeled transition system is the usual one (in the early style). As an example, here are the rules for input, and one of the communication rules:

$$\frac{}{p(x).\,P \xrightarrow{pv} P\{x \to v\}} \qquad \frac{P \xrightarrow{\overline{a}(b)} P' \quad Q \xrightarrow{ab} Q'}{P \mid Q \xrightarrow{\tau} (\nu\, b)(P' \mid Q')}$$

where $b \notin \mathrm{fn}(Q)$. *Actions*, ranged over by μ, can be of four forms: τ (interaction), pq (an input at p in which q is received), $\overline{p}q$ (free output) and $\overline{p}(q)$ (bound output). In these actions, p is the *subject*. Free and bound names of actions and processes are defined as usual. Relation $\Longrightarrow$ is the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\overset{\mu}{\Longrightarrow}$ stands for $\Longrightarrow \xrightarrow{\mu} \Longrightarrow$. The class of *image-finite processes* is the largest subset $\mathcal{I}$ of processes which is derivation closed and s.t. $P \in \mathcal{I}$ implies that, for all μ, the set $\{P' \ : \ P \overset{\mu}{\Longrightarrow} P'\}$, quotiented by alpha conversion, is finite.

The rules for subtyping and typing are in Figure 2. Note that type annotation i (input capability) gives covariance, o (output capability) gives contravariance, and b (both capabilities) gives invariance. *Type environments,* ranged over by Γ and Δ, are finite assignments of types to names. A typing judgement $\Gamma \vdash P$ asserts that process P is well-typed in Γ, and $\Gamma \vdash a : T$ that name a has type T in Γ. We write $\Gamma \leq \Delta$ if whenever $\Delta \vdash a : T$ then $\Gamma \vdash a : T$. In the rules for input and output prefixes, the subject of the prefix is checked to possess the appropriate input or output capability in the type environment. There is only one rule, for name typing, which explicitly uses subtyping. This type system enjoys expected properties of type systems (like subject reduction and narrowing [15]).

3 Typed barbed equivalence

The behavioural equality we adopt for the π-calculus is *barbed equivalence.* Barbed equivalence has been used for a broad variety of calculi; its main advantage is that it can be uniformly defined on different calculi, for it requires of the calculus little more than a notion of reduction – the τ-step of the π-calculus. Barbed equivalence is defined on top of *barbed bisimulation*, using a universal quantification on contexts.

We refine the standard notions of barbed bisimulation and equivalence by adding an (external) type environment and a typed notion of observable. P and Q barbed equivalent w.r.t. a typing Δ means, intuitively, that the two processes are undistinguishable under the assumption that the observer's behaviour respects the types in Δ. Comparing typed processes and a typed observer makes sense only if these types are *compatible*; that is, there is a type environment under which both the processes and the observer are well-typed. (For simplicity, we assume that the free names of the processes are all known to the observer.)

Definition 3.1 (compatibility) *The relation* $\mathbf{K}_{\mathcal{P}}$ *of* compatible triples *is defined as*

$$\mathbf{K}_{\mathcal{P}} \overset{\mathrm{def}}{=} \{(P, \Delta, Q) \ : \ \textit{there is } \Gamma \textit{ with } \Gamma \vdash P, Q \textit{ and } \Gamma \leq \Delta \textit{ and } \mathrm{fn}(P,Q) \subseteq \mathrm{domain}(\Delta)\}.$$

For any $\mathcal{R} \subseteq \mathbf{K}_{\mathcal{P}}$ and any type environment Δ, we define $\mathcal{R}_\Delta \overset{\mathrm{def}}{=} \{(P,Q) \ : \ (P,\Delta,Q) \in \mathcal{R}\}$. We write $P \downarrow a$ (resp. $P \downarrow \overline{a}$) if $P \xrightarrow{\mu} P'$ for some P' and input (resp. output) action μ with subject a; and $P \Downarrow a$ (resp. $P \Downarrow \overline{a}$) if $P \Longrightarrow \downarrow a$ (resp. $P \Longrightarrow \downarrow \overline{a}$).

Definition 3.2 (typed barbed equivalence) *Let $\mathcal{R} \subseteq \mathbf{K}_{\mathcal{P}}$ and symmetric. We say that $\mathcal{R}$ is a* typed barbed bisimulation *if for all P, Q and Δ it holds that $P \ \mathcal{R}_\Delta \ Q$ implies, for all names x:*

1. *if $\Delta \vdash x : T^{\mathsf{o}}$, for some T, and $P \downarrow x$ then $Q \Downarrow x$;*
2. *if $\Delta \vdash x : T^{\mathsf{i}}$, for some T, and $P \downarrow \overline{x}$ then $Q \Downarrow \overline{x}$, and*
3. *if $P \xrightarrow{\tau} P'$ then there exists Q' s.t. $Q \Longrightarrow Q'$ and $P' \ \mathcal{R}_\Delta \ Q'$*

We say that P and Q are barbed bisimilar w.r.t. Δ, *written $\Delta \rhd P \overset{\bullet}{\approx} Q$, if $P \ \mathcal{R}_\Delta \ Q$ for some typed barbed bisimulation $\mathcal{R}$. We say that P and Q are* barbed equivalent w.r.t. Δ, *written $\Delta \rhd P \simeq Q$, if $\Delta \rhd P \mid R \overset{\bullet}{\approx} Q \mid R$, for all R with $\Delta \vdash R$.*

In the untyped case, barbed equivalence coincide with the ordinary (early) labeled bisimilarity (on image-finite processes). One of the main goals of this paper is to find a similar purely-coinductive characterisation of barbed equivalence in the typed case.

4 A labelled transition system

Our characterization of barbed equivalence relies on a new labelled transition system (LTS) for the typed π-calculus (Figure 3). The states of this LTS record some information on the types that the observer has on names. As explained in the Introduction, these types impose a constrain on the actions of processes that an observer may test (by contrast, in a untyped setting, the observer is allowed unconstrained usage of the names). We also need to keep track that the observer

Names			
	$a, b, c, \ldots p, q, r, \ldots x, y, z$		
Types			
T	$::=$	T^I	capability type
	$\mid$	$\bullet$	unit
Capability tags			
I	$::=$	i	input only
	$\mid$	o	output only
	$\mid$	b	either

Processes			
P	$::=$	$\mathbf{0}$	nil process
	$\mid$	$P \mid P$	parallel
	$\mid$	$(\nu\, x\,)P$	restriction
	$\mid$	$p(x).\, P$	input
	$\mid$	$\overline{p}v.\, P$	output
	$\mid$	$!P$	replication
	$\mid$	$P + P$	summation

Figure 1: The syntax of the typed π-calculus

Subtyping rules:

$$\frac{}{S \le S} \qquad \frac{I \in \{\mathsf{b}, \mathsf{i}\} \qquad S \le T}{S^I \le T^{\mathsf{i}}} \qquad \frac{I \in \{\mathsf{b}, \mathsf{o}\} \qquad T \le S}{S^I \le T^{\mathsf{o}}} \qquad \frac{}{S \le \bullet}$$

Process typing:

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad \frac{\Gamma \vdash P}{\Gamma \vdash !P} \qquad \frac{\Gamma, x : S^{\mathsf{b}} \vdash P}{\Gamma \vdash (\nu\, x\,)P}$$

$$\frac{}{\Gamma \vdash \mathbf{0}} \qquad \frac{\Gamma \vdash p : S^{\mathsf{i}} \qquad \Gamma, x : S \vdash P}{\Gamma \vdash p(x).\, P} \qquad \frac{\Gamma \vdash p : S^{\mathsf{o}} \qquad \Gamma \vdash w : S \qquad \Gamma \vdash P}{\Gamma \vdash \overline{p}w.P}$$

Name typing:

$$\frac{\Gamma(p) = T}{\Gamma \vdash p : T} \qquad \frac{\Gamma \vdash v : S \qquad S \le T}{\Gamma \vdash v : T}$$

Figure 2: Subtyping and typing rules for the π-calculus

may receive a name several times, with different types. For instance, let $Q \stackrel{\mathrm{def}}{=} (\nu\, b)(\overline{c}b.\overline{c'}b.c''(x))$ and suppose that the observer knows c, c' and c'' with types $T^{\mathsf{o}^{\mathsf{i}}}$, $T^{\mathsf{i}^{\mathsf{i}}}$ and $T^{\mathsf{b}^{\mathsf{o}}}$ (for some T), respectively. Q can communicate its private name b to the observer twice with types T^{o} and T^{i}. The observer cannot send b at c'', because the type of c'' requires that the observer has the b-capability on b. However, if the observer had a type $T^{\mathsf{b}^{\mathsf{i}}}$ on c, then a process $(\nu\, b)(\overline{c}b.c''(x))$ could send b at c to the observer, and then the observer could send b back at c''. These two examples show that receiving a copy of a name with the input capability and another copy with the output capability is different from (it is more constraining than) receiving just one copy with both capabilities. To allow observers with multiple types on names, our LTS records the type information for the observer in *typed closures.* These are pairs $[\Delta, \rho]$, where Δ is a type environment and ρ a substitution. Names in Δ represent occurrences which are mapped to "real" names by ρ. A *configuration* is a closure $[\Delta, \sigma]$ together with a π-calculus process P, written $[\Delta,\ \sigma]\,\#\,P$. The set of all configurations is denoted by $\mathcal{C}$ and is ranged over by H, K. In a configuration, the observer is represented by the closure; observer and process must be compatible, that is, there is a type environment under which both are well-typed.

Definition 4.1 (typed closures) *A* typed closure *is a pair* $[\Delta, \rho]$ *with* $\mathrm{domain}(\Delta) = \mathrm{domain}(\rho)$. *We write* $[\Delta, \rho] \vdash a : T$ *if, for some* x, *we have* $\Delta \vdash x : T$ *and* $x\rho = a$. *Moreover, given two typed closures* $[\Gamma, \sigma]$ *and* $[\Delta, \rho]$, *we write* $[\Gamma, \sigma] \leq [\Delta, \rho]$ *if whenever* $[\Delta, \rho] \vdash a : T$ *then* $[\Gamma, \sigma] \vdash a : T$.

For a typing Δ, we write ϵ_Δ for the substitution that is the identity on $\mathrm{domain}(\Delta)$, and is undefined elsewhere. Note that $\Gamma \leq \Delta$ iff $[\Gamma, \epsilon_\Gamma] \leq [\Delta, \epsilon_\Delta]$.

Definition 4.2 (configurations) *We shall write* $[\Delta,\ \sigma]\ \#\ P$, *and call it a* configuration, *if* $[\Delta, \sigma]$ *is a typed closure,* $\mathrm{fn}(P) \subseteq \mathrm{codomain}(\sigma)$ *and there is a* Γ *s.t.* $\Gamma \vdash P$ *and* $[\Gamma, \epsilon_\Gamma] \leq [\Delta, \sigma]$.

Note that the existence of the type environment Γ guarantees that σ preserves types of names in Δ, up to subtyping. We define now transitions on configurations. A transitions

$$[\Delta,\ \sigma]\,\#\,P \xrightarrow[\delta]{\alpha} [\Delta',\ \sigma']\,\#\,P'$$

represents an internal activity of the system composed of the process in parallel with the observer, in which the process contributes α and the observer contributes $\delta\sigma'$. We call α the *process action*, $\delta\sigma'$ the observer's *real action*, and δ the observer's *virtual action* (the process action and the observer's real action are the dual of each other—they must give rise to an interaction). The grammars for α and δ are:

$$\begin{array}{lcl} \alpha & := & - \mid \mu \\ \delta & := & - \mid x(v) \mid \overline{x}z \mid \overline{x}(v) \mid \langle \mu, \lambda \rangle\,. \end{array}$$

Symbol "$-$" occurs when the process (in the case of α) or the observer (in the case of δ) is idle—it does not contribute to the transition. Action $\langle \mu, \lambda \rangle$ represents an internal move of the observer, in which a subcomponent contributes μ and another subcomponent contributes λ; for this interaction to happen, the real actions corresponding to μ and λ must be the dual of each other. The new LTS is defined by the rules in Figure 3 . In `F-Out`, the observer sends a global name $z\sigma$ at $x\sigma$, and the process receives it; for this to happen, Δ must posses the appropriate capabilities on x and z (condition $\Delta \vdash \overline{x}z$). In rule `F-Inp`, the observer receives a name b with a type T; a new entry is added to the closure with value b and type T. The last two rules model internal transitions of the observer. Rule `F-Com` says that if two distinct name x and y in the closure are aliases for the same name ($x\sigma = y\sigma$) and have dual capabilities, then an internal transition can take place, where the sender transmits a name $z\sigma$ with type T, and the recipient can use it with a type S (by the compatibility condition on configurations, T must be a subtype of S). A new entry is added to the closure to record the substitution of names that this communication has produced in the observer. Rule `B-Com` is similar, but here the communicated name is a private (hence new) name w.

The notations used for process transitions are extended to configuration transitions. For instance,

$$H \Longrightarrow H' \text{ stands for } H(\xrightarrow[-]{\tau})^* H',$$

$$H \overset{\alpha}{\underset{\delta}{\Longrightarrow}} H' \text{ stands for } H \Longrightarrow \xrightarrow[\delta]{\alpha} \Longrightarrow H'.$$

and

$$H \overset{\widehat{\alpha}}{\underset{\delta}{\Longrightarrow}} H'$$

stands for $H \overset{\alpha}{\underset{\delta}{\Longrightarrow}} H'$ if $\alpha \neq \tau$ and for $H \Longrightarrow H'$ otherwise. For any two binary relations $\rightarrow_1$ and $\rightarrow_2$, we write $A \rightarrow_1 \diamond \rightarrow_2 B$ to mean that either $A \rightarrow_1 \rightarrow_2 B$ or $A \rightarrow_2 \rightarrow_1 B$ hold.

5 Bisimulation with capabilities

In the untyped labeled bisimulation, two processes are bisimilar if they can perform the *same* actions, and

In these rules, we assume that w and v are fresh names.

$$(\texttt{Tau})\ \frac{P \xrightarrow{\tau} P'}{[\Delta,\ \sigma]\# P \xrightarrow[-]{\tau} [\Delta,\ \sigma]\# P'}$$

$$(\texttt{F-Out})\ \frac{P \xrightarrow{ab} P' \quad x\sigma = a \quad z\sigma = b \quad \Delta \vdash \overline{x}z}{[\Delta,\ \sigma]\# P \xrightarrow[\overline{x}z]{ab} [\Delta,\ \sigma]\# P'} \qquad (\texttt{B-Out})\ \frac{P \xrightarrow{aw} P' \quad x\sigma = a \quad \Delta \vdash x : T^{\mathsf{o}}}{[\Delta,\ \sigma]\# P \xrightarrow[\overline{x}(w)]{aw} [\Delta_{[w:T]},\ \sigma_{[w\to w]}]\# P'}$$

$$(\texttt{F-Inp})\ \frac{P \xrightarrow{\overline{a}b} P' \quad x\sigma = a \quad \Delta \vdash x : T^{\mathsf{i}}}{[\Delta,\ \sigma]\# P \xrightarrow[x(w)]{\overline{a}b} [\Delta_{[w:T]},\ \sigma_{[w\to b]}]\# P'} \qquad (\texttt{B-Inp})\ \frac{P \xrightarrow{\overline{a}(w)} P' \quad x\sigma = a \quad \Delta \vdash x : T^{\mathsf{i}}}{[\Delta,\ \sigma]\# P \xrightarrow[x(w)]{\overline{a}(w)} [\Delta_{[w:T]},\ \sigma_{[w\to w]}]\# P'}$$

$$(\texttt{F-Com})\ \frac{x \neq y \quad x\sigma = y\sigma \quad \Delta \vdash x : T^{\mathsf{o}} \quad \Delta \vdash z : T \quad \Delta \vdash y : S^{\mathsf{i}}}{[\Delta,\ \sigma]\# P \xrightarrow[\langle \overline{x}z,\ y(w)\rangle]{-} [\Delta_{[w:S]},\ \sigma_{[w\to z\sigma]}]\# P}$$

$$(\texttt{B-Com})\ \frac{x \neq y \quad x\sigma = y\sigma \quad \Delta \vdash x : T^{\mathsf{o}} \quad \Delta \vdash y : S^{\mathsf{i}}}{[\Delta,\ \sigma]\# P \xrightarrow[\langle \overline{x}(w),\ y(v)\rangle]{-} [\Delta_{[w:T,v:S]},\ \sigma_{[w\to w,\ v\to w]}]\# P}$$

Figure 3: Operational semantics of configurations

recursively so on their derivatives. If the two processes perform the same actions, then also the real actions of an observer interacting with them are the same. Therefore also the virtual actions of the observer are the same, because in the untyped case virtual and real actions coincide. We take bisimulation to mean that *in matching transitions the observer's virtual actions should be the same.* We define bisimulation on configurations accordingly. As a result, in matching transitions of our typed bisimulation, the observer's real actions, and hence also the process actions, may be different.

It only makes sense to compare configurations that have the same observer and a common typing.

Definition 5.1 (compatibility on configurations) *The relation $\mathbf{K}_{\mathcal{C}}$ of compatible configurations is so defined*

$$\mathbf{K}_{\mathcal{C}} \stackrel{\mathrm{def}}{=} \Big\{ \big([\Delta,\ \sigma]\# P,\ [\Delta,\ \rho]\# Q\big) \ : \textit{there is } \Gamma \textit{ with } \Gamma \vdash P, Q \textit{ and } [\Gamma, \epsilon_\Gamma] \leq [\Delta, \sigma] \textit{ and } [\Gamma, \epsilon_\Gamma] \leq [\Delta, \rho] \Big\}.$$

Given $\mathcal{S} \subseteq \mathbf{K}_{\mathcal{C}}$ and fixed any Δ, σ and ρ, we write $[\sigma\ ,\ \Delta,\ \rho] \rhd P\ \mathcal{S}\ Q$ if $([\Delta,\ \sigma]\# P)\ \mathcal{S}\ ([\Delta,\ \rho]\# Q)$.

Definition 5.2 (typed bisimulation) *Let $\mathcal{S} \subseteq \mathbf{K}_{\mathcal{C}}$ and symmetric. We say that $\mathcal{S}$ is a* typed bisimulation *if $H\mathcal{S}K$ implies (bound names of actions are assumed to be fresh):*

1. *if $H \xrightarrow[\delta]{\alpha} H'$, with $\alpha \neq -$, then there are β and K' such that*

$$K \overset{\widehat{\beta}}{\underset{\delta}{\Longrightarrow}} K' \textit{ and } H'\mathcal{S}K';$$

2. *if $H \xrightarrow[\langle\mu,\ \lambda\rangle]{-} H'$ then there is K' s.t. either:*
   (a) $K \overset{-}{\underset{\langle\mu,\ \lambda\rangle}{\Longrightarrow}} K'$ and $H'\mathcal{S}K'$, or
   (b) for some α and β, $K \overset{\alpha}{\underset{\mu}{\Longrightarrow}} \diamond \overset{\beta}{\underset{\lambda}{\Longrightarrow}} K'$ and $H'\mathcal{S}K'$

We say that H and K are typed bisimilar, *written $H \approx K$, if there is a bisimulation $\mathcal{S}$ with $H\mathcal{S}K$.*

The labels α, β (those on top of arrows) are not used in the bisimulation; we give them to help reading the clauses. In Clause (2), the step from H represents an internal move of the observer. K may match this move in the same way (clause 2.a); but it may also match it via some interactions between the observer and the process (clause 2.b). The latter may happen when the closure in H has an aliasing that the closure in K does not have. See Examples 2 and 3 in Section 7 for uses of this clause.

5.1 Techniques of "bisimulation up–to"

Labeled bisimulation represents a powerful proof technique for proving typed process equalities. To make it even more powerful we enhance it with *up–to techniques* [11, 19]. Given a functional $\mathcal{F} : \mathcal{P}(\mathbf{K}_{\mathcal{C}}) \longrightarrow \mathcal{P}(\mathbf{K}_{\mathcal{C}})$, a *typed bisimulation up to* $\mathcal{F}$ is defined like a typed bisimulation (Definition 5.2), but with the condition "$H'\mathcal{S}K'$" replaced by the (weaker) "$H'\mathcal{F}(\mathcal{S})K'$". Different functionals correspond to different up–to techniques. A technique up to $\mathcal{F}$ is *sound* if any bisimulation up to $\mathcal{F}$ is included in $\approx$. Up–to techniques are useful to reduce both the size and the number of the configurations to consider when proving bisimilarities, as one is allowed to match the transitions of the two processes being compared up to some transformations (represented by $\mathcal{F}$) on their derivatives. For lack of space, below we introduce some useful sound up–to techniques only informally. These three techniques allow us to discard entries of closures, to narrow types, and to discard restrictions in processes:

- in bisimulation *up to weakening*, given a derivative of the form $[\Delta_{[x:T]},\ \sigma_{[x \to a]}] \# P$ (where Δ and σ are not defined on x), we can discard $x : T$ and $[x \to a]$ if name a does not appear in P.
- in bisimulation *up to narrowing*, given a derivative $[\Delta,\ \sigma] \# P$ we can replace $[\Delta, \sigma]$ with $[\Delta', \sigma']$ if $[\Delta', \sigma'] \leq [\Delta, \sigma]$.
- in bisimulation *up to restriction*, given a a pair of derivatives $(\ [\Delta,\ \sigma] \# (\nu c)P,\ \ [\Delta,\ \rho] \# Q\ )$, we can open the restriction on c in one of the configurations and continue with a pair $(\ [\Delta_{[x:U]},\ \sigma_{[x\to c]}] \# P,\ \ [\Delta_{[x:U]},\ \rho_{[x \to b]}] \# Q\ )$, for some b and U.

Furthermore, any combination of the listed up–to techniques (formally, any composition of the corresponding functionals) is still a sound up–to technique.

We can also adapt up–to techniques known for untyped bisimulations; for instance, bisimulation up to bisimulation, bisimulation up to expansion, bisimulation up to parallel composition. In our typed setting, bisimulation up to parallel composition allows us to cut parallel components of the form $R\sigma$ and $R\rho$ in pairs of derivatives of the form $(\ [\Delta,\ \sigma] \# P \mid R\sigma,\ [\Delta,\ \rho] \# Q \mid R\rho\ )$, provided that $\Delta \vdash R$.

6 Soundness and completeness w.r.t. barbed equivalence

In this section we prove that our labeled bisimulation coincides with barbed equivalence on the class of image-finite processes. For establishing soundness, we have to prove that the labeled bisimulation is preserved by parallel composition. This is, technically, quite different and harder than the proofs of analogous results for ordinary bisimilarities.

Lemma 6.1 *If* $[\Gamma, \epsilon_\Gamma] \leq [\Delta, \sigma]$ *and* $\Delta \vdash R$ *then* $\Gamma \vdash R\sigma$.

Lemma 6.2 (parallel composition) *If* $[\sigma\ ,\ \ \Delta,\ \ \rho] \rhd P \ \approx\ Q$ *and* $\Delta \vdash R$ *then* $[\sigma\ ,\ \Delta,\ \rho] \rhd P \mid R\sigma \ \approx\ Q \mid R\rho$.

PROOF: By showing that relation (this relation is contained in $\mathbf{K}_{\mathcal{C}}$ by Lemma 6.1)

$$\Big\{\big([\Delta,\ \sigma] \# P \mid R\sigma,\ [\Delta,\ \rho] \# Q \mid R\rho\big) : \\ [\sigma,\ \Delta,\ \rho] \rhd P \ \approx\ Q \text{ and } \Delta \vdash R\Big\}$$

is a bisimulation up to bisimulation, restriction and narrowing. In this proof, the up–to techniques are very important; it would be very hard to give the full bisimulation relations otherwise. $\square$

We abbreviate $[\epsilon_\Delta,\ \Delta,\ \epsilon_\Delta] \rhd P \ \approx\ Q$ to $\Delta \rhd P \approx Q$.

Corollary 6.3 (soundness) *If* $\Delta \rhd P \approx Q$ *then* $\Delta \rhd P \simeq Q$.

The proof of completeness, that follows the proof schema for untyped bisimulations, relies on: (i) a characterization of $\approx$ on configurations whose process is image-finite given as the intersection of all the inductive approximants $\approx^n$, $n \geq 0$, and (ii) the existence of certain test processes $R(n, \Delta)$, such that $\Delta \rhd P \mid R(n, \Delta)\sigma \ \ \dot{\approx}\ \ Q \mid R(n, \Delta)\rho$ implies $[\sigma\ ,\ \Delta,\ \rho] \rhd P \ \approx^n\ Q$.

Theorem 6.4 (completeness) *Let* P *and* Q *be image-finite processes. If* $\Delta \rhd P \simeq Q$ *then* $\Delta \rhd P \approx Q$.

7 Examples

Example 1 Consider the processes $P \overset{\text{def}}{=} \overline{b}(x).a(y).(y \mid \overline{x})$ and $Q \overset{\text{def}}{=} \overline{b}(x).a(y).(y.\overline{x} + \overline{x}.y)$. As explained in the Introduction, these processes are not bisimilar in the ordinary π-calculus. It is simple to prove that they are bisimilar w.r.t. a type environment Δ s.t. $\Delta \vdash a : T^{\mathsf{i}^{\mathsf{o}}}$ and $\Delta \vdash b : T^{\mathsf{o}^{\mathsf{i}}}$.

Example 2 Let $P \overset{\text{def}}{=} \overline{c}a.R$ and $Q \overset{\text{def}}{=} \overline{c}b.R$, with $R \overset{\text{def}}{=} !a(z).\,\overline{b}z \,|\, !b(z).\,\overline{a}z$. These processes are not bisimilar in the untyped π-calculus because their initial transitions have different labels, namely $\overline{c}a$ and $\overline{c}b$. We can prove they are bisimilar w.r.t. any type environment Δ in which $\Delta \vdash c : S^{\mathsf{i}}$ does not hold for any S. For the proof, we can use relation $\mathcal{S}$ composed of the two pairs

$$([\Delta,\ \epsilon_\Delta]\#P,\ [\Delta,\ \epsilon_\Delta]\#Q) \quad \text{and}$$

$$([\Delta_{[v:T]},\ \epsilon_{\Delta[v\to a]}]\#R,\ [\Delta_{[v:T]},\ \epsilon_{\Delta[v\to b]}]\#R)$$

for T s.t. $\Delta(c) = T^I$. This relation is a typed bisimulation up to parallel composition and narrowing.

In this proof, clause 2.b of the definition of typed bisimulation may be needed. We briefly explain why. Suppose that, for some S, $\Delta \vdash c : S^{\mathsf{oi}}$ and $\Delta(a) = \Delta(b) = S^{\mathsf{i}}$. The action

$$[\Delta,\ \epsilon_\Delta]\#P \xrightarrow[c(v)]{\overline{c}a} ([\Delta_{[v:S^{\mathsf{o}}]},\ \epsilon_{\Delta[v\to a]}]\#R) \overset{\text{def}}{=} H$$

is matched by

$$[\Delta,\ \epsilon_\Delta]\#Q \xrightarrow[c(v)]{\overline{c}b} ([\Delta_{[v:S^{\mathsf{o}}]},\ \epsilon_{\Delta[v\to b]}]\#R) \overset{\text{def}}{=} K.$$

This creates in the closure of H an aliasing between two names a and v (both mapped onto a) with complementary capabilities (v has type S^{o}, and a type S^{i}). We can now use rule `B-Com` to infer

$$H \xrightarrow[\langle \overline{v}(z),\, a(w)\rangle]{-} H',$$

for some H'. To match this transition, K uses clause (2.b) and makes two communications with the link $!b(z).\overline{a}z.$ in R, thus:

$$K \xrightarrow[\overline{v}(z)]{b(z)} \xrightarrow[a(w)]{\overline{a}z} K'.$$

Note that (H', K') belongs to $\mathcal{S}$ up to narrowing.

Example 3 Suppose, for some T, $\Delta \vdash a : T^{\mathsf{oi}}$ and $\Delta \vdash b : T^{\mathsf{i}}$. Due to clause 2 of the definition of bisimulation, processes $\overline{a}b$ and $\overline{a}(x)$ are not Δ-bisimilar. In fact, $[\Delta_{[x:T^{\mathsf{o}}]},\ \epsilon_{\Delta[x\to b]}]\#\mathbf{0}$ has a transition of the form

$$[\Delta_{[x:T^{\mathsf{o}}]},\ \epsilon_{\Delta[x\to b]}]\#\mathbf{0} \xrightarrow[\langle \overline{x}z,\, bz\rangle]{-} H$$

for some z and H, that $[\Delta_{[x:T^{\mathsf{o}}]},\ \epsilon_{\Delta[x\to x]}]\#\mathbf{0}$ cannot match. Indeed, the two processes could not be distinguished without the help of clause 2.

Example 4: replication theorem. Capability types have been introduced in [15]. There, the main application of types to behavioural equivalence of processes is a stronger version of Milner's replication theorem. The assertion of the theorem reads thus: a passive resource R that is shared among a certain number of clients can be made private to each of them. In our notation, it becomes:

Theorem 7.1 *Suppose that*

(a) $\Gamma, a : S^{\mathsf{o}} \vdash P_1 \mid P_2$, *and*

(b) $\Gamma, a : S^{\mathsf{o}}, z : S \vdash R$, *for some* z.

Then $\Gamma \rhd (\nu\, a)(\,!\,a(z).R \mid P_1 \mid P_2) \approx (\nu\, a)(\,!\,a(z).R \mid P_1) \mid (\nu\, a)(\,!\,a(z).R \mid P_2)$.

We can give a much simpler proof than that in [15] using our proof techniques based on the typed labeled bisimulation. We take the relation consisting of the pairs

$$\begin{aligned}(\ &[\Gamma',\ \epsilon_{\Gamma[a\to a, a'\to a]}]\#\ !\,a(z).R \mid P\{a' \to a\},\\ &[\Gamma',\ \epsilon_{\Gamma'}]\#\ !\,a(z).R \mid\ !\,a'(z).R\{a \to a'\} \mid P\ \ )\end{aligned}$$

for $\Gamma' \overset{\text{def}}{=} \Gamma, a : S^{\mathsf{o}}, a' : S^{\mathsf{o}}$, and Γ s.t. $\Gamma \vdash P$, and $\Gamma, a : S^{\mathsf{o}}, z : S \vdash R$. This relation is a typed bisimulation up to bisimulation, parallel composition, weakening and narrowing. The thesis then follows from simple algebraic manipulations (congruence laws for restrictions) and taking P to be $P_1 \mid (P_2\{a' \to a\})$.

The stronger replication theorem can be used, for instance, to prove an optimisation in Milner's encoding of call-by-value λ-calculus into the π-calculus.

Example 5: functions. This example uses communication of tuples and of integers, which are straightforward to accommodate in the theory developed in the previous sections. We wish to prove the correctness of implementations of functions as mobile processes. Consider a service $P \overset{\text{def}}{=} !\, f(m,s).\ \overline{s}\langle \mathit{fact}(m)\rangle$ that, when passed a natural m and a return address s at channel f, gives back the factorial of m at s. Consider now an implementation of the same function where factorial is computed in the usual recursive way, relying on the operations of subtraction and multiplication:

$$\begin{aligned}Q \overset{\text{def}}{=} !\, f(m,s).\ &\texttt{if}\ m = 0\ \texttt{then}\ \overline{s}\langle 1\rangle\ \texttt{else}\\ &(\nu\, s')(\overline{f}\langle m-1, s'\rangle.s'(v).\overline{s}\langle m * v\rangle)\end{aligned}$$

We expect that P and Q are behaviourally equivalent, but they are not in the untyped case, as Q can do an action $(\nu\, s')\overline{f}\langle m-1, s'\rangle$ after $f(m,s)$, whereas P cannot. Intuitively, an untyped observer can interfere with the recursive call of Q at $\overline{f}$, thus disrupting the underlying protocol of Q. However, we can prove that P and Q are equivalent, provided the environment obeys the typing discipline for f given by $\Delta \overset{\text{def}}{=} \{f : \langle \mathbf{N}, \mathbf{N}^{\mathsf{b}}\rangle^{\mathsf{o}}\}$. Note, in particular, that the environment cannot use f for input.

Theorem 7.2 $\Delta \rhd P \approx Q$.

PROOF: By showing that the relation consisting of the *single* pair $\big([\Delta,\ \epsilon_\Delta] \# P,\ [\Delta,\ \epsilon_\Delta] \# Q\ \big)$ is a typed bisimulation up to expansion and parallel composition. $\square$

Other examples of applications of our typed bisimulation can be found in [8], where we use them in π-calculus interpretations of Abadi and Cardelli's calculi of objects to prove properties of objects involving typing and subtyping.

8 Related work and extensions

Pierce and Sangiorgi [16] study behavioural equivalences in a π-calculus with polymorphic types. Only barbed equivalence is studied (no labeled bisimulation), but the distinction is made between typing of the observer and typing of the processes, which has inspired our work. Hennessy and Riely [4] study a language with an explicit notion of locations and primitives on them; they have richer set of capabilities, and a richer subtyping relation than us. Only contextual forms of behavioural equivalences are given. Characterisations of barbed equivalence on calculi for mobile processes include Boreale, Laneve and Fournet [2], Amadio [1] and Sangiorgi [21]. However, in these bisimilarities, matching transitions of processes have the same labels (either because there is matching on all names, or because all names communicated are private), therefore the problems of aliasing dealt with in this paper do not appear. Other studies of barbed equivalence, or similar contextual-based notions of bisimulation, for typed mobile processes include [6, 7, 25]. Merro and Sangiorgi [10] study barbed equivalence and congruence in an asynchronous π-calculus where only the output capability of names may be exported. The direct characterisations given are simpler than ours, but they only work under both the hypothesis of asynchrony and of output-capability-only. Other works related to ours are [9],[12] and [14]. In [9], Larsen and Xinxin introduce *action transducers*, where interaction between processes and the surrounding contexts is made explicit: this is somehow reminiscent of our LTS for typed bisimulation. Mason and Talcott in [14] and Montanari and Pistore in [12] discuss reasons for requiring or excluding, respectively, forms of matching in the considered process calculi.

As for untyped bisimulations, closing our labeled bisimilarity under name substitutions gives us a characterisation of typed barbed *congruence* (the congruence induced barbed bisimilarity—with quantification over all contexts).

The results in this paper are also useful in untyped calculi without matching. This corresponds to having only the b-capability. In this case, all typing information in configurations can be ignored. For example, using the labeled bisimilarity we can give a simple proof that in the untyped π-calculus without matching $\overline{a}b.\,(!\overline{b} \mid !b \mid !\overline{c} \mid !c)$ is equivalent to $\overline{a}c.\,(!\overline{b} \mid !b \mid !\overline{c} \mid !c)$.

The results are also useful in typed calculi with the matching construct. We may for instance add a testing capability on names or, as suggested in [15], allow that a name passed with both input and output capability may be tested for equality. If two configurations $[\Delta,\ \sigma] \# P$ and $[\Delta,\ \rho] \# Q$ are bisimilar and a name x appears in Δ with the testing capabilities, then we should require that the values $\sigma(x)$ and $\rho(x)$ be the same (the observer can look at them). However, the types in Δ still impose constraints on the actions the observer can offer.

Our results can be extended to other calculi with first-order types and subtyping on them. For instance, calculi with variants, records, products, and recursive types (with recursive types, some additional care is needed when proving completeness for barbed equivalence, depending on the definition of type equality). We do not know how to extend these results to calculi with higher-order types; higher-order types introduce new problems—see [16].

It should be possible to extend our results to asynchronous calculi [5] but clause (2) of the definition of labeled bisimulation might become more complex.

Acknowledgments

We are grateful to Benjamin Pierce for several discussions on typed calculi for mobile processes. The anonymous referees provided useful suggestions and comments.

References

[1] R. Amadio. An asynchronous model of locality, failure, and process mobility. In Proc. Coordination'97, volume 1282 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[2] M. Boreale, C. Fournet, and C. Laneve. Bisimulations for the Join Calculus. To appear in *Proc. PROCOMET'98*.

[3] C. Fournet and G. Gonthier The Reflexive Chemical Abstract Machine and the Join calculus. In *Proc. 23th POPL*. ACM Press, 1996.

[4] M. Hennessy and J. Riely. A typed language for distributed mobile processes. In *Proc. 25th POPL*. ACM Press, 1997.

[5] K. Honda and M. Tokoro. On asynchronous communication semantics. *ECOOP '91* 1991 , volume 612 of *Lecture Notes in Computer Science*, Springer Verlag, 1992.

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

[7] N. Kobayashi, B.C. Pierce, and D.N. Turner. Linearity and the pi-calculus. In *Proc. 23th POPL*. ACM Press, 1996.

[8] J. Kleist and D. Sangiorgi. Imperative objects and mobile processes. To appear in *Proc. PROCOMET'98*.

[9] K. Larsen and L. Xinxin. Compositionality through an Operational Semantics of Contexts. Journal of Logic and Computation 1(6):761–793, 1991.

[10] M. Merro and D. Sangiorgi. The asynchronous π-calculus, revisited. To appear in *Proc. of ICALP'98*.

[11] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[12] U. Montanari and M. Pistore. Checking Bisimilarity for Finitary π-calculus. In *Proc. of CONCUR'95*, 1995.

[13] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.

[14] I. Mason and C. Talcott. A Semantically Sound Actor Translation. In *Proc. of ICALP'97*, 1997.

[15] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Journal of Mathematical Structures in Computer Science*, 6(5):409–454, 1996. An extended abstract in *Proc. LICS 93*, IEEE Computer Society Press.

[16] B. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. In *24th POPL*. ACM Press, 1997.

[17] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. Technical Report CSCI 476, Indiana University, 1997. To appear in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, MIT Press.

[18] J. C. Reynolds. Preliminary design of the programming language Forsythe. Tech. rept. CMU-CS-88-159. Carnegie Mellon University., 1988.

[19] D. Sangiorgi. On the proof method for bisimulation. *Proc. MFCS'95*, volume 969 of *Lecture Notes in Computer Science*, pages 479–488. Springer Verlag, 1995.

[20] D. Sangiorgi. An interpretation of typed objects into typed π-calculus. Technical Report RR-3000, INRIA-Sophia Antipolis, 1996. To appear in *Information and Computation*.

[21] D. Sangiorgi. The name discipline of receptiveness. In *24th ICALP*, volume 1256 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[22] B. Victor and F. Moller. The Mobility Workbench — a tool for the π-calculus. *Proc. CAV'94*, volume 818 of *Lecture Notes in Computer Science*, Springer-Verlag, 1994.

[23] B. Victor and J. Parrow. Constraints as processes. In *Proc. CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.

[24] D. Walker. Objects in the π-calculus. *Information and Computation*, 116(2):253–271, 1995.

[25] N. Yoshida. Graph types for monadic mobile processes. In *Proc. FST & TCS*, volume 1180 of *Lecture Notes in Computer Science*, pages 371–386. Springer Verlag, 1996.