The name discipline of uniform receptiveness (extended abstract)

Davide Sangiorgi

INRIA - Sophia Antipolis, France.

1 Introduction

The π-calculus [9] is a paradigmatical process calculus for message-passing concurrency. Two processes with acquaintance of a given name can use it to interact with each other. Names themselves may be exchanged in communications, which can model modifications of the linkage structure among processes. These are the basic process constructs (using lower case for names and upper case for processes): $\overline{a}\langle b\rangle.\, P$, the output of b at a with P as continuation; $a(b).\, P$, an input at a with b placeholder for the name received in the input; $P_1 \mid P_2$, the parallel composition of the two processes; $\boldsymbol{\nu} a\, P$, which makes name a local to P; and $!P$, which denotes a potentially-infinite number of copies of P in parallel.

In this paper, we study the situation in which certain names are *uniformly receptive*. A name x is receptive in a process P if at any time P is able of offering an input at x (at least as long as there are processes that could send messages at x). The receptiveness of x is uniform if all inputs at x have the same continuation. Receptiveness ensures that any message sent at x can be immediately processed; unformity ensures that there is a unique way in which a message at x may be processed (that is, the input end of x is "functional").

These are *semantic* conditions, and are undecidable. To obtain decidable conditions we impose some restrictions. Roughly, we guarantee receptiveness by demanding that the name is available in input-replicated form as soon as created. For instance, x is receptive in

$$P_1 \overset{\text{def}}{=} \boldsymbol{\nu} x\, (!x(p).\, P \mid Q) \qquad P_2 \overset{\text{def}}{=} \boldsymbol{\nu} x\, (\overline{r}\langle x\rangle.\, !x(p).\, P) \tag{1}$$

(On the right, name x is created when the output $\overline{r}\langle x\rangle$ is consumed since, before this, x is frozen.[1]) We guarantee uniformity by demanding that there is only one input occurrence of the name; hence in (1) name x should not occur free in input position in P and Q. To preserve the uniformity property in a network of processes, we then also demand that only the output capability of the name may be transmitted; that is, as all π-calculus names, so uniform receptive names can be transmitted but, in contrast with the other names, they can be used by a recipient only in output (retransmitting the name, or sending a message at it).

In the processes P_1 and P_2 above, the receptiveness at x is persistent, which is necessary if unboundedly many messages could be sent at x. It is useful to

---

[1] Indeed P_2 is behaviourally the same as $\boldsymbol{\nu} x\, (!x(p).\, P \mid \overline{r}\langle x\rangle)$, which is of the same form as P_1.

consider separately the case in which at most one message can be sent. Then the replication in front of the input at x is unnecessary. We call the first form *ω-receptiveness*, the second *linear receptiveness.*

Uniform receptiveness corresponds to a precise discipline in the usage of names; it could by formulated by syntactic means, but it is easier and more elegant to do so using a type system along the lines of type systems for the π-calculus. The impact of receptiveness on behavioural equivalences and process reasoning is the main focus of this paper. We shall develop some theory and proof techniques for processes with receptive names, and then illustrate their usefulness by means of some non-trivial examples, like the proof of some transformations that introduce parallelism in a resource, and the proof of the correctness of an optimisation of the translation of higher-order process calculi into the π-calculus [18, 13], which is adopted in the compiler of Pict [12].

The challenge in these examples is that the equalities implied by the transformations fail in the ordinary π-calculus (even w.r.t. the very coarse notion of trace equivalence). That is, there are contexts of the ordinary π-calculus that are able to detect the difference between the processes of the equalities. By imposing the type system for receptiveness, these contexts are ruled out as ill-typed.

Uniform receptiveness often occurs in the π-calculus. Our first example is the coding of functions. A process Q with a local function $\lambda r.\, M$, accessible via a name z, is normally written $\boldsymbol{\nu} z\,(!z(r,y).\, P \mid Q)$ where P is the coding of M and y is (a placeholder for) the name where the result of a function call will be delivered. Within Q, a call of the function with argument n is written $\boldsymbol{\nu} x\,(\overline{z}\langle n,x\rangle.\, x(p).\, Q')$ where p is (a placeholder for) the result of the call. In the function declaration, z is ω-receptive; in the function call, x is linear receptive. Similar combinations of linear and ω-receptiveness occur in the coding of higher-order communications and of Object-Oriented languages. Typically, ω-receptiveness occurs in the modelling of resources which are private to one or more client processes (above, the resource is a function). A discipline similar to ω-receptiveness is presently used in the compiler of Pict [12], to allow optimisations of the code implementing communications. An important example of linear receptiveness (indeed, perhaps the most important) is found in process interactions based on the *Remote Procedure Call* (RPC) paradigm. An RPC interaction involves two synchronisations between a caller and a callee where, after the first synchronisation, the caller waits the time necessary for the callee to elaborate a response. When we are modeling RPC's in the π-calculus, the return name at which the callee delivers its response is used as linear receptive. (The function call above too is an example of an RPC interaction.)

As behavioural equivalence on processes, we use *barbed equivalence.* This equates processes which, very roughly, in all contexts give rise to the same patterns of interactions. The main inconvenience of barbed equivalence is that it uses quantification over contexts in the definition, and this can make proofs of processes equality heavy. Against this, one looks for *direct* characterisations, without context quantification. For instance, in CCS and in the ordinary π-calculus barbed equivalence coincides with the well-known *labeled* bisimilarities

[13]. (In a labeled bisimilarity the bisimulation game is played not only on silent actions, as for barbed bisimulation, but also on input and output actions.)

We sketch the essential points of our theory for processes with receptive names. The schema is the same for linear and for ω receptiveness. We first introduce a type system which forces the receptiveness discipline, and prove some basic properties for it. Secondly, we isolate a subclass of the well-typed processes, called *discreet processes*, roughly characterised by the property that all receptive names which are emitted are private to the sender. Discreet processes are defined by means of syntactic restrictions on the output prefix similar to those in the language πI [15]. Thirdly, we introduce a simple but powerful algebraic law, with which any well-typed process can be transformed into a discreet process. Remarkably, this law equates a process whose first action is the output of a *global* name with a process whose first action is the output of a *private* name. The law is not valid in the untyped π-calculus, but it is valid under the receptiveness type system. Finally, we prove a direct characterisation of barbed equivalence on discreet processes, as a labelled bisimilarity called *receptive bisimilarity.* The latter differs from the ordinary bisimilarity in the requirement for input actions, but otherwise it can be used with the standard co-inductive techniques of labelled bisimilarities, including proof techniques such as "bisimulation up to expansion".

For lack of space, some definitions and most of the proofs are omitted. More examples can be found in [17].

2 Some background on the π-calculus

We use lower case letters $p, q, r, \ldots$ to range over names, and upper case letters P, Q, R to range over the set $\mathcal{P}$ of processes. This is the π-calculus grammar (for simplicity, we develop our theory on the monadic calculus):

$$\begin{array}{rl} P := 0 & \big|\; p(q).\,P \;\big|\; \overline{p}\langle q\rangle.\,P \;\big|\; \overline{p}(q).\,P \;\big|\; [p=q]P \\ & \big|\; P_1 \mid P_2 \;\big|\; \boldsymbol{\nu} p\, P \;\big|\; P_1 + P_2 \;\big|\; !p(q).\,P \end{array}$$

We allow the bound-output prefix $\overline{p}(q).\,P$ in the syntax; often in the π-calculus literature, $\overline{p}(q).\,P$ is given as an abbreviation for $\boldsymbol{\nu} q\, \overline{p}\langle q\rangle.\,P$. 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. We write $\overline{p}.\,P$ and $p.\,P$ when the name transmitted at p is not important, and we often abbreviate $\alpha.\,0$ as α. The labeled transition system is the usual one, in the early style. *Actions*, ranged over by μ, can be of four forms: τ (interaction), $p\langle q\rangle$ (an input at p in which q is received), $\overline{p}\langle q\rangle$ (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. In a statement, we say that a name is *fresh* to mean that it is different from the names of other processes or actions in the statement. Relation $\Longrightarrow$ is the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\overset{\mu}{\Longrightarrow}$ stands for $\Longrightarrow \xrightarrow{\mu} \Longrightarrow$. $P \Downarrow_p$ holds if there is P' and an action μ with subject p s.t. $P \overset{\mu}{\Longrightarrow} P'$. A context C is *static* if it has the form $\boldsymbol{\nu}\widetilde{p}\,(P \mid [\cdot])$, for some P and $\tilde{p}$.

Definition 1 barbed bisimulation, equivalence and congruence. *Barbed bisimulation* is the largest symmetric relation $\stackrel{\bullet}{\approx}$ on processes s.t. $P \stackrel{\bullet}{\approx} Q$ implies:

1. whenever $P \Longrightarrow P'$ then there exists Q' such that $Q \Longrightarrow Q'$ and $P' \stackrel{\bullet}{\approx} Q'$;
2. for each name p, $P \Downarrow_p$ iff $Q \Downarrow_p$.

Two processes P and Q are *barbed equivalent*, written $P \approx Q$, if for each static context C it holds that $C[P] \stackrel{\bullet}{\approx} C[Q]$; they are *barbed congruent*, written $P \simeq Q$, if $C[P] \stackrel{\bullet}{\approx} C[Q]$ for all contexts.

Barbed equivalence and congruence usually coincide with the ordinary labeled (early) bisimilarity and congruence of the π-calculus [13]. The proof of this fact is simple on the class of the image finite processes (to which most of the processes one would like to write belongs) by exploiting the n-approximants of the labeled equivalences. We recall that the class of *image-finite processes* is the largest subset $\mathcal{I}$ of $\mathcal{P}$ which is derivation closed and s.t. $P \in \mathcal{I}$ implies that, for all μ, the set $\{P' \ : \ P \stackrel{\mu}{\Longrightarrow} P'\}$, quotiented by α conversion, is finite.

3 Linear receptiveness

The discipline of uniform receptiveness (briefly receptiveness) can be added to any of the main existing type systems for the π-calculus. In this paper, our base type system will be Milner's *sorting*, that we now briefly recall. Names are partitioned into a collection of *sorts*. Then a sorting function is defined which maps sorts onto sorts (in the polyadic calculus it maps sorts onto sequences of sorts). If a sort s is mapped onto a sort t this means that names in s may only carry names in t; moreover, t is the *object sort* of s. In the remainder, we shall assume that there is a sorting system under which all processes are well-typed. We separate the base type system (Milner's sorting) from the typing rules for receptiveness so as to show the essence of the latter rules.

We begin our analysis of receptiveness from the case of *linear receptiveness*. We call the non-linear-receptive names *plain names*. There are no constraints on plain names except those imposed by the underlying sorting. We shall omit the adjective "linear" when there is no ambiguity. For simplicity, we assume that: There is a single sort `L-recep` of linear receptive names; linear receptive names carry plain names. These two assumptions can be relaxed without difficulties. We also assume the existence of a sort `trig` of names, different from `L-recep` but with the same object sort as `L-recep` (note that names in `trig` are plain names). The sort `trig` will be used to derive simpler characterisations of our bisimilarities. *In the remainder, $x, y, z \ldots$ range over linear receptive names, $a, b, \ldots$ over plain names, and v over names in* `trig`. *We recall that p, q, r range over the set of all names. Δ, Γ range over finite sets of linear receptive names.* We sometimes write $\Delta - x$ as abbreviation for $\Delta - \{x\}$ and Δ, x for $\Delta \cup \{x\}$, and also x for $\{x\}$. The type system for linear receptiveness is in Table 3. A rule with double conclusion is an abbreviation for more rules with same premises but separate conclusions. Judgements have the form $\Delta; \Gamma \vdash P$. As sets, the order in

$$\texttt{(T-inp-mat)}\ \frac{\emptyset;\Gamma\vdash P}{\emptyset;\Gamma\vdash a(b).\,P,\ [a=b]P} \qquad \texttt{(T-inp-2)}\ \frac{x\notin\Gamma \qquad \emptyset;\Gamma,x\vdash P}{\emptyset;\Gamma\vdash a(x).\,P}$$

$$\texttt{(T-inp-3)}\ \frac{x\notin\Gamma \qquad \emptyset;\Gamma\vdash P}{x;\Gamma\vdash x(b).\,P} \qquad \texttt{(T-rep)}\ \frac{\emptyset;\emptyset\vdash a(b)P}{\emptyset;\emptyset\vdash !a(b).\,P}$$

$$\texttt{(T-res-1)}\ \frac{x\notin\Gamma\cup\Delta \qquad \Delta,x;\Gamma,x\vdash P}{\Delta;\Gamma\vdash \boldsymbol{\nu}x\,P} \qquad \texttt{(T-res-2)}\ \frac{p\notin\Gamma\cup\Delta \qquad \Delta;\Gamma\vdash P}{\Delta;\Gamma\vdash \boldsymbol{\nu}p\,P}$$

$$\texttt{(T-out-1)}\ \frac{\emptyset;\Gamma\vdash P}{\emptyset;\Gamma\vdash \overline{a}\langle b\rangle.\,P,\ \ \overline{a}(b).\,P} \qquad \texttt{(T-out-2)}\ \frac{x\notin\Gamma \qquad \emptyset;\Gamma\vdash P}{\emptyset;\Gamma,x\vdash \overline{a}\langle x\rangle.\,P}$$

$$\texttt{(T-out-3)}\ \frac{x\notin\Gamma \qquad \emptyset;\Gamma\vdash P}{\emptyset;\Gamma,x\vdash \overline{x}\langle b\rangle.\,P,\ \ \overline{x}(b).\,P} \qquad \texttt{(T-bout)}\ \frac{x\notin\Gamma \qquad x;\Gamma\vdash P}{\emptyset;\Gamma\vdash \overline{a}(x).\,P}$$

$$\texttt{(T-par)}\ \frac{\Delta_1;\Gamma_1\vdash P_1 \qquad \Delta_2;\Gamma_2\vdash P_2 \qquad \Delta_1\cap\Delta_2=\emptyset \qquad \Gamma_1\cap\Gamma_2=\emptyset}{\Delta_1,\Delta_2;\Gamma_1,\Gamma_2\vdash P_1 \mid P_2}$$

$$\texttt{(T-nil)}\ \frac{}{\emptyset;\emptyset\vdash 0} \qquad \texttt{(T-sum)}\ \frac{\emptyset;\Gamma\vdash P_1 \qquad \emptyset;\Gamma\vdash P_2}{\emptyset;\Gamma\vdash P_1+P_2}$$

Table 1. Typing rules for linear receptiveness.

which names appear in Δ and Γ does not matter. Intuitively, if $\Delta;\Gamma\vdash P$ then $\Delta\cup\Gamma$ are the only receptive names which appear free in P; process P must use any name in Γ *exactly once* in output position (that is, either performing an output at that name or transmitting this capability to another process), and names in Δ *immediately* and *only once* in input. This intuition is formalised in Theorem 2, which relates types and operational semantics of processes. We say that P is *well typed* if there are Δ,Γ s.t. $\Delta;\Gamma\vdash P$ holds.

Theorem 2 soundness theorem. *Suppose* $\Delta;\Gamma\vdash P$.

1. *if* $x\in\Delta$ *then for all* a *there is a unique* P' *s.t.* $P\xrightarrow{x\langle a\rangle}P'$*;*
2. *If* $P\xrightarrow{a\langle x\rangle}P'$ *and* $x\notin\Gamma$ *then* $\Delta;\Gamma,x\vdash P'$*;*
3. *if* $P\xrightarrow{x\langle a\rangle}P'$ *then* $x\in\Delta$ *and* $\Delta-x;\Gamma\vdash P'$*;*
4. *if* $P\xrightarrow{a\langle b\rangle}P'$ *or* $P\xrightarrow{\overline{a}\langle b\rangle}P'$ *or* $P\xrightarrow{\overline{a}(b)}P'$*, then* $\Delta;\Gamma\vdash P'$*;*
5. *if* $P\xrightarrow{\tau}P'$ *then either* $\Delta;\Gamma\vdash P'$ *or there is* $x\in\Delta\cap\Gamma$ *and* $\Delta-x;\Gamma-x\vdash P'$*;*
6. *if* $P\xrightarrow{\overline{x}\langle a\rangle}P'$ *or* $P\xrightarrow{\overline{x}(a)}P'$ *or* $P\xrightarrow{\overline{a}\langle x\rangle}P'$*, then* $x\in\Gamma$ *and* $\Delta;\Gamma-x\vdash P'$*;*
7. *if* $P\xrightarrow{\overline{a}(x)}P'$ *and* $x\notin\Delta\cup\Gamma$ *then* $\Delta,x;\Gamma\vdash P'$.

Behavioural equivalences under linear receptiveness As usual in typed calculi, the definitions of the barbed relations take typing into account, so that the composition of a context and a process be well-typed. With receptiveness, an additional ingredient has to be taken into account, namely the input availability of receptive names. If a process has the possibility of using certain receptive names in output, then a context in which the process is tested should guarantee the input-availability at these names, otherwise the essence of receptiveness — outputs at receptive names can be immediately consumed — is lost.

Definition 3 complete processes and contexts. A process P is *complete* if $\Delta; \emptyset \vdash P$, for some Δ. We say that context C is *complete on $(\Delta;\Gamma)$* if $C[P]$ is complete, for all P s.t. $\Delta;\Gamma \vdash P$.

Definition 4 barbed equivalences under linear receptiveness. Suppose $\Delta;\Gamma \vdash P, Q$. Then we say that P and Q are *barbed equivalent under linear receptiveness at* $(\Delta;\Gamma)$, briefly $P \approx_{\mathtt{L}}^{\Delta;\Gamma} Q$, if for each static context C which is complete on $(\Delta;\Gamma)$ it holds that $C[P] \stackrel{\centerdot}{\approx} C[Q]$ (where $\stackrel{\centerdot}{\approx}$ is barbed bisimulation, Definition 1). *Barbed congruence under linear receptiveness at* $(\Delta;\Gamma)$, briefly $\simeq_{\mathtt{L}}^{\Delta;\Gamma}$, is defined similarly — just remove the constraint on C being static.

We write $\Delta;\Gamma \vdash_D P$ if $\Delta;\Gamma \vdash P$ can be proved without using rule `T-out-2`; in this case we say that P is *discreet*. In a discreet process, all receptive names which are exported must be private: Syntactically, this means that outputs of global receptive names are disallowed (that is, using the terminology in [15], only *internal mobility* — the sending of fresh names — is allowed on receptive names). We write $p \rhd q$ as abbreviation for a process $p(r).\, \overline{q}\langle r\rangle.\, 0$ (a 1-place ephemeral buffer from p to q). We can transform well-typed processes into discreet processes using the law

$$\overline{b}\langle x\rangle.\, P = \overline{b}(y).\, (y \rhd x \mid P) \qquad \text{for } y \text{ fresh} \tag{2}$$

This law makes the output of a *global* name into the output of a *local* (i.e., private) name. The law is not valid in the ordinary π-calculus, but it is valid under receptiveness:

Lemma 5. *If $\Delta;\Gamma \vdash \overline{b}\langle x\rangle.\, P$ and y is fresh, then $\overline{b}\langle x\rangle.\, P \simeq_{\mathtt{L}}^{\Delta;\Gamma} \overline{b}(y).\, (y \rhd x \mid P)$.*

We now derive a characterisation of the receptive barbed equivalence as a labeled bisimulation on discreet processes. We begin by defining the labeled bisimilarity on *complete* discreet processes. We say that an action μ is a *plain input* if μ is the input of a plain name, i.e., $\mu = p\langle a\rangle$ for some plain name a.

Definition 6 linear-receptive bisimilarity, $\asymp_{\mathtt{L}}$. *Linear-receptive bisimilarity* is the largest relation $\asymp_{\mathtt{L}}$ on complete discreet processes s.t. $P \asymp_{\mathtt{L}} Q$ implies:

1. if $P \xrightarrow{\mu} P'$ with bound name of μ (if it exists) fresh for P and Q, and μ is an output or a plain input then there is Q' s.t. $Q \stackrel{\mu}{\Longrightarrow} Q'$ and $P' \asymp_{\mathtt{L}} Q'$;
2. if $P \xrightarrow{\tau} P'$ then there is Q' s.t. $Q \Longrightarrow Q'$ and $P' \asymp_{\mathtt{L}} Q'$;
3. if $P \xrightarrow{p\langle x\rangle} P'$ and x is fresh for P and Q then, for some fresh name v, there are Q' and Q'' s.t.: (a) $Q \stackrel{p\langle x\rangle}{\Longrightarrow} Q'$; (b) $\boldsymbol{\nu} x\, (x \rhd v \mid Q') \Longrightarrow Q''$; (c) $\boldsymbol{\nu} x\, (x \rhd v \mid P') \asymp_{\mathtt{L}} Q''$.

The main novelty of receptive bisimulation is the use of a process $x \triangleright v$ in the input clause (3). To understand this addition, recall that x represents a private receptive name that the observer exports; if the observer behaves as a well-typed process, then it must make x immediately available in input, as a process of the form $x(p).\, R$. It is perhaps surprising that we do not test the behaviour of the derivatives P' and Q' for all infinite choices of the process $x(p).\, R$, but only on a single, simple, process, namely a link $x \triangleright v$.

Definition 7 linear-receptive bisimilarity on all discreet processes. Suppose $\varDelta; \varGamma \vdash_D P, Q$. Let $\widetilde{x} = \varDelta \cap \varGamma$ and $\widetilde{y} = \varGamma - \varDelta$ (therefore $\varGamma = \widetilde{x} \cup \widetilde{y}$); and let $\widetilde{v}$ be fresh and pairwise distinct names with $\mid \widetilde{y} \mid = \mid \widetilde{v} \mid$.

We then set $P \asymp_{\mathrm{L}}^{\varDelta;\varGamma} Q$ if $(\boldsymbol{\nu} \widetilde{x}, \widetilde{y}\,)(\widetilde{y} \triangleright \widetilde{v} \mid P) \asymp_{\mathrm{L}} (\boldsymbol{\nu} \widetilde{x}, \widetilde{y}\,)(\widetilde{y} \triangleright \widetilde{v} \mid Q)$.

The definition makes sense because processes $(\boldsymbol{\nu} \widetilde{x}, \widetilde{y})(\widetilde{y} \triangleright \widetilde{v} \mid P)$ and $(\boldsymbol{\nu} \widetilde{x}, \widetilde{y})(\widetilde{y} \triangleright \widetilde{v} \mid Q)$ are complete and discreet, and we have already defined $\asymp_{\mathrm{L}}$ on this class. Moreover, since on complete processes $\asymp_{\mathrm{L}}$ is preserved by structural equality and injective renaming, the above definition does not depend on the order of names in $\widetilde{x}, \widetilde{y}$ and $\widetilde{v}$, and on the choice of names $\widetilde{v}$.

The closure of barbed bisimulation w.r.t. the static contexts gives the ordinary (early) labeled bisimulation [13]; the closure w.r.t. the *complete* static contexts gives receptive bisimulation. The proofs for the ordinary bisimulation can be adapted to receptive bisimulation. Here are further useful laws for receptive barbed equivalence that are easy to prove using the labeled bisimilarity $\asymp_{\mathrm{L}}^{\varDelta;\varGamma}$, and that are not valid in the ordinary π-calculus:

$$\text{If } \varDelta; \varGamma \vdash \overline{x}\langle p \rangle.\, P, \text{ then } \overline{x}\langle p \rangle.\, P \approx_{\mathrm{L}}^{\varDelta;\varGamma} \overline{x}\langle p \rangle \mid P. \tag{3}$$

Suppose that $\varDelta; \varGamma \vdash P, Q$, for some $\varDelta$ and $\varGamma$ with $x \in \varDelta - \varGamma$, and let v be a fresh name; then

$$P \approx_{\mathrm{L}}^{\varDelta;\varGamma} Q \quad \text{iff} \quad \boldsymbol{\nu} x\, (v \triangleright x \mid P) \approx_{\mathrm{L}}^{\varDelta - x;\varGamma} \boldsymbol{\nu} x\, (v \triangleright x \mid Q) \tag{4}$$

Suppose that $\varDelta; \varGamma \vdash P, Q$, for some $\varDelta$ and $\varGamma$ with $y \in \varGamma - \varDelta$, and let v be a fresh name; then

$$P \approx_{\mathrm{L}}^{\varDelta;\varGamma} Q \quad \text{iff} \quad \boldsymbol{\nu} y\, (y \triangleright v \mid P) \approx_{\mathrm{L}}^{\varDelta;\varGamma - y} \boldsymbol{\nu} y\, (y \triangleright v \mid Q). \tag{5}$$

Law (3) transforms a "synchronous" output into an "asynchronous" one; (4) transforms a global input into a local input; (5) does the same for outputs.

4 $\boldsymbol{\omega}$-receptiveness

The other interesting example of uniform receptiveness is *ω-receptiveness*, where: The input of a name is always available, and always with the same continuation; there are no limitations on the utilisation of the name in output. A simple way of ensuring the uniformity condition on inputs is to require that the only input occurrence be replicated, i.e., of the form $!x(p).\, P$.

When adapting the theory of linear receptiveness to ω-receptiveness, there are several, but not surprising, modifications to make. In the typing system, the interpretation of a judgement $\Delta; \Gamma \vdash P$ is now that P must make names in Δ immediately available, in input-replicated form; whereas it may use names in Γ arbitrarily many times in output. We only show the new version of rules `T-par` and `T-out-2`, and one of the rules for replication:

$$\frac{\Delta_1; \Gamma \vdash P_1 \qquad \Delta_2; \Gamma \vdash P_2 \qquad \Delta_1 \cap \Delta_2 = \emptyset}{\Delta_1, \Delta_2; \Gamma \vdash P_1 \mid P_2}$$

$$\frac{x \in \Gamma \qquad \emptyset; \Gamma \vdash P}{\emptyset; \Gamma \vdash \overline{a}\langle x\rangle.\, P} \qquad\qquad \frac{\emptyset; \Gamma \vdash P}{x; \Gamma \vdash\, !x(b).\, P}$$

In the definitions of the typed barbed relations, typed labeled bisimilarities and the algebraic laws for the ω-case, the main modification w.r.t. the linear case is that the links $p \rhd q$ have to become persistent. Using $\simeq_\omega^{\Delta;\Gamma}$ for barbed congruence under ω-receptiveness at $\Delta; \Gamma$, law (2) becomes

$$\overline{b}\langle x\rangle.\, P \simeq_\omega^{\Delta;\Gamma} \overline{b}(y).\, (!y \rhd x \mid P) \qquad \text{for } y \text{ fresh} \tag{6}$$

5 Examples

Parallelisation of resources We can use linear receptiveness to validate transformations that increase the parallelism in processes. In the processes below, we use recursion, polyadicity and communication of integers, which are straightforward to accommodate in the theory of bisimulation previously developed (recursion can be coded up). Thus m, n range over integers and variables over integers. Consider the process:

$$A_1\langle b\rangle \overset{\text{def}}{=} a(x).\, b(n, c).\, \boldsymbol{\nu} d\, \overline{c}\langle d\rangle.\, \overline{x}\langle n\rangle.\, A_1\langle d\rangle$$

A client can interrogate $A_1\langle b\rangle$ at a, and it will receive at the return channel x an integer n that $A_1\langle b\rangle$ has received at another channel b (this channel is renewed at each cycle using c). Interactions between A_1 and the clients are Remote Procedure Calls (RPC), therefore the return channels are used according to the discipline of linear receptiveness (see the discussion on RPC in the introductory section). The behaviour of A_1 is strictly sequential. Let us introduce some parallelism:

$$A_2\langle b\rangle \overset{\text{def}}{=} a(x).\, b(n, c).\, \boldsymbol{\nu} d\, \Big(\overline{x}\langle n\rangle.\, \overline{c}\langle d\rangle \mid A_2\langle d\rangle\Big)$$

$$A_3\langle b\rangle \overset{\text{def}}{=} a(x).\, \boldsymbol{\nu} d\, \Big(b(n, c).\, \overline{c}\langle d\rangle.\, \overline{x}\langle n\rangle \mid A_3\langle d\rangle\Big)$$

Process $A_2\langle b\rangle$ can accept a second request at a before the answer to the fist request has been delivered; however answers cannot overtake one another — they are delivered in the same order in which the requests were made. Process $A_3\langle b\rangle$ can even accept a request before receiving an integer at b; answers can overtake. Let now $I\langle n\rangle$ be a counter $I\langle n, b\rangle \overset{\text{def}}{=} \boldsymbol{\nu} c\, \overline{b}\langle n, c\rangle.\, c(d).\, I\langle n+1, d\rangle$ and consider the systems (n is any integer) $\mathtt{S}i\langle n\rangle \overset{\text{def}}{=} \boldsymbol{\nu} b\, (A_i\langle b\rangle \mid I\langle n, b\rangle)$, for $i \in \{1, 2, 3\}$.

All these systems are distinguished in the ordinary π-calculus — the different degrees of parallelism that they exhibit are observable. We can prove that they *are* equivalent exploiting the linear receptiveness of the return channels x, y. [2] For $\mathtt{S1}\langle n\rangle \approx_{\mathtt{L}}^{\emptyset;\emptyset} \mathtt{S2}\langle n\rangle$ one proves that the relation composed by all pairs of the form

$$\Big(\mathtt{S1}\langle n'\rangle \mid \prod_{i=1}^{m} \overline{v_i}\langle n_i\rangle, \mathtt{S2}\langle n'\rangle \mid \prod_{i=1}^{m} \overline{v_i}\langle n_i\rangle\Big)$$

for some channel v_i and integers m, n_i, n' is a $\asymp_{\mathtt{L}}$-bisimulation up to expansion. The other equalities can be proved in a similar way.

The above processes are simple. A more interesting example of parallelisation of resources is Cliff Jones's parallelisation transformation problem [5]. We analyse this in [16], where we prove Jones's transformation using a combination of the techniques for linear and ω receptiveness.

Encoding of higher-order process calculi We now present an example with ω-receptiveness. Below x, y *are supposed to be* ω*-receptive names.* We prove the correctness of an optimisation of the translation of higher-order process calculi into the π-calculus [13, 18]. In a higher-order calculus, terms of the languages may be transmitted. For simplicity of presentation, we consider the simpler case of a calculus where *only* processes may be communicated. The operators are those for sending a process ($\overline{p}\langle P_1\rangle . P_2$), receiving a process ($p(X). P$), process variable ($X$), plus the usual operators of restriction, parallel composition, summation, replication. This calculus, which we call HOPC, is the core of Plain CHOCS [18], and is a second-order fragment of the Higher-Order π-calculus [13]. Upper case letter X ranges over process variables. A process is *closed* if it does not contain free variables. The compilation $\mathcal{C}$ of this calculus into the π-calculus in [13, 18] acts as a homomorphism on all process constructs except input, output prefixes and process variables where it is so defined:

$$\begin{array}{ll} \mathcal{C}[\![\overline{p}\langle P\rangle . Q]\!] \stackrel{\mathrm{def}}{=} \boldsymbol{\nu} x\, \overline{p}\langle x\rangle . (!x. \mathcal{C}[\![P]\!] \mid \mathcal{C}[\![Q]\!]) & \text{for } x \text{ fresh} \\ \mathcal{C}[\![p(X). Q]\!] \stackrel{\mathrm{def}}{=} p(x). \mathcal{C}[\![Q]\!]) & \mathcal{C}[\![X]\!] \quad \stackrel{\mathrm{def}}{=} \quad \overline{x}. 0 \end{array}$$

In the compilation, the communication of a process P is translated as the communication of a private name which acts as a pointer to (the translation of) P and which the recipient can use to trigger a copy of (the translation of) P. These pointers, introduced in the compilation, are used as ω-receptive names.

In [13], the correctness of compilation $\mathcal{C}$ is established, by proving that it is fully abstract w.r.t. barbed congruence (that is, for all closed HOPC processes P and Q, $P \simeq Q$ iff $\mathcal{C}[\![P]\!] \simeq \mathcal{C}[\![Q]\!]$). The optimisation that we consider acts on outputs of process variables. Let us call $\mathcal{O}$ the optimised compilation. It is defined as $\mathcal{C}$ except for the case of an output of a variable, for which we have:

$$\mathcal{O}[\![\overline{p}\langle X\rangle . Q]\!] \stackrel{\mathrm{def}}{=} \overline{p}\langle x\rangle . \mathcal{O}[\![Q]\!]$$

---

[2] In these definitions, also name c is linear receptive. We do not need this fact for the proofs (and it is reasonable not to use it, because the linear receptiveness of c is accidental — one can modify the definitions so that c is not linear receptive.)

For instance, when translating $p(X).\,\overline{q}\langle X\rangle.\,0$, the result of $\mathcal{O}$ is $p(x).\,\overline{q}\langle x\rangle.\,0$ while that of $\mathcal{C}$ is $p(x).\,\boldsymbol{\nu} y\,\overline{q}\langle y\rangle.\,!y.\,\overline{x}.\,0$. The optimisation avoids us one level of indirection through pointers. This optimisation is analysed in [13] and is shown to be unsound for untyped barbed equivalence. However, we can show that the optimisation is sound if we take into account the receptiveness of names. The proof is an immediate consequence of law (6), since, for all P, $\mathcal{O}[\![P]\!]$ can be transformed into $\mathcal{C}[\![P]\!]$ by repeatedly applying the law:

Theorem 8. *Let P be a HOPC process with free variables in $\{X_1, \ldots, X_n\}$, and let $\Gamma \overset{\mathrm{def}}{=} \{x_1, \ldots, x_n\}$. It holds that $\mathcal{C}[\![P]\!] \simeq_{\omega}^{\emptyset;\Gamma} \mathcal{O}[\![P]\!]$.*

Combining this with the theorem on $\mathcal{C}$ in [13], we can prove: for all closed HOPC processes P and Q, $P \simeq Q$ iff $\mathcal{O}[\![P]\!] \simeq_{\omega}^{\emptyset;\emptyset} \mathcal{O}[\![Q]\!]$.

In an expanded paper [17], other examples of application of ω-receptiveness are reported: The proof of the equivalence between the target processes of Milner's two encodings of call-by-values λ-calculus into π-calculus [8] (this is a novel result); the proofs of some stronger versions of π-calculus replication theorems [10] (these results were already proved in [10]; exploiting receptiveness we get simpler proofs).

6 Final remarks

Several type systems have been proposed for process calculi. The most relevant for this work are [10], where the type system has input/output modalities to distinguish between the capabilities of reading and writing on names, and the type systems expressing linearity information [3, 7, 4]. The type system for receptiveness represents a refinement of [10] and, in the case of linear receptiveness, also of [7]. Also [10] and [7] contain studies of the effect of types on process behaviours, using barbed congruence. The proof techniques developed in this paper are easier to apply, mainly because based on labeled bisimilarities.

Other papers with results on behavioural consequences of π-calculus types include the following. [6] defines a type system for the asynchronous π-calculus that guarantees deadlock freedom in certain cases; a subsystem of this system is similar to ours for ω-receptiveness. [19] uses a type system where types have a graph structure to prove the full abstraction of an encoding of the polyadic π-calculus into the monadic calculus. Graphs allow expressing sophisticated communication protocols but introduce some complications in the typing. [14] uses a type system with input/output modalities and variant types to guarantee the adequacy of a translation of a typed object-oriented calculus into the π-calculus. [11] studies the constraints imposed by parametric polymorphism.

Some of the ideas in this paper should be useful to develop reasoning techniques for other type systems, in particular those with input/output modalities and with linearity. They might also be useful in cases where either the receptiveness or the uniformity condition fails; for instance the calculus in [2], where all names are uniform but not necessarily receptive, or that in [1], where all names are receptive but not necessarily uniform.

Acknowledgements. The author would like to thank G. Boudol, N. Kobayashi, C. Jones, B. Pierce, D. Walker, N. Yoshida and two of the anonymous referees for useful discussions or suggestions. This research has been supported by FRANCE TÉLÉCOM, CTI-CNET 95-1B-182, "Modélisation de Systèmes Mobiles".

References

1. R. Amadio. Locality and failures II. To appear as a Technical Report, INRIA-Sophia Antipolis, 1997.
2. Fournet C. and Gonthier G. The Reflexive Chemical Abstract Machine and the Join calculus. In *Proc. 23th POPL*. ACM Press, 1996.
3. K. Honda. Types for dydadic interaction. In E. Best, editor, *Proc. CONCUR '93*, LNCS 715, pages 509–523. Springer Verlag, 1993.
4. K. Honda. Composing processes. In *Proc. 23th POPL*. ACM Press, 1996.
5. C.B. Jones. Constraining interference in an object-based design method. *Proc. TAPSOFT'93*, LNCS 668, pages 136–150. Springer Verlag, 1993.
6. N. Kobayashi. A partially deadlock-free typed process calculus. To appear as a technical report, Department of Information Science, University of Tokyo, 1997. A summary is to appear in *Proc. 12th LICS Conf.*, 1997.
7. N. Kobayashi, B.C. Pierce, and D.N. Turner. Linearity and the pi-calculus. In *Proc. 23th POPL*. ACM Press, 1996.
8. R. Milner. Functions as processes. Research Report 1154, INRIA, Sophia Antipolis. Final version in *J. Mathem. Struct. in Computer Science* 2(2):119–141, 1992.
9. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.
10. B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Journal of Mathematical Structures in Computer Science*, 6(5):409–454, 1996.
11. B. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. In *24th POPL*. ACM Press, 1997.
12. B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In preparation, 1997.
13. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, University of Edinburgh, 1992.
14. D. Sangiorgi. An interpretation of typed objects into typed π-calculus. Technical Report RR-3000, INRIA-Sophia Antipolis, 1996.
15. D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(2):235–274, 1996.
16. D. Sangiorgi. Typed π-calculus at work: a proof of Jones's parallelisation transformation on concurrent objects. Presented at Fourth Workshop on Foundations of Object-Oriented Languages, 1997. Available as `ftp://zenon.inria.fr/meije/theorie-par/davides/JonesTransform.ps.gz`.
17. D. Sangiorgi. The name discipline of uniform receptiveness. (This is a slightly longer version of the present paper.) Available as `ftp://zenon.inria.fr/meije/theorie-par/davides/RecePi.ps.Z`.
18. B. Thomsen. Plain CHOCS, a second generation calculus for higher-order processes. *Acta Informatica*, 30:1–59, 1993.
19. Nobuko Yoshida. Graph types for monadic mobile processes. In *Proc. FST & TCS*, LNCS 1180, pages 371–386. Springer Verlag, 1996.